Proyecto de Sistemas Informáticos
Facultad de Informática
Universidad Complutense de Madrid

Simulación de confianza y reputación.

Sistema multi-agente de una cadena de mercados.

Jose Alberto Contreras Fernández

Carlos Eiras Franco

Sergio Garrote Merino

Profesor director: Juan Pavón Mestras

Palabras para búsqueda bibliográfica:

confianza, reputación, agente, mercado, simulación, Java, Repast y MercaMadrid.

Los alumnos que hemos desarrollado este proyecto autorizamos a la Universidad Complutense de Madrid a difundir y a utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la propia memoria como el código, la documentación y/o el prototipo desarrollado.

José Alberto Contreras Fernández **Carlos Eiras Franco** **Sergio Garrote Merino**

ÍNDICE

Resumen

En esta memoria describimos todo el trabajo realizado como proyecto para la asignatura Sistemas Informáticos de la Ingeniería Superior en Informática de la Universidad Complutense de Madrid. Planteamos un sistema multi-agente basado en la confianza y reputación que realiza una simulación de una cadena de mercados. Para la mejor comprensión del lector se ha creado un apartado de documentación en esta memoria donde describimos en términos generales los conceptos de confianza y reputación y simulación con agentes. El caso de estudio elegido ha sido Mercamadrid y se explicará más adelante los detalles de adaptación del sistema. También hemos incluido una descripción detallada de la estructura y comportamiento del programa así como las herramientas que este proporciona. Por último, exponemos un estudio sobre los resultados de las simulaciones y las conclusiones obtenidas del año de trabajo y orientamos a los interesados en continuar este trabajo.

Abstract

In this document, all the work that has been done as a project for the subject Sistemas Informáticos *(Computing Engineering Complutense University) is described.* We propose a Multiagent System based on trust and reputation which runs a simulation of a market chain. For your better comprehension we have included a documentation section in which we describe the concepts of both Trust and reputation in Multiagent systems and Agent simulation. We have chosen Mercamadrid as our object to study and the details about the adaptation of this market to our system will be explained later. Furthermore, we have included a detailed description of the structure and behaviour of the program as well as the tools provided. Finally, we put forward a study about the results of the simulations and the conclusions obtained and we, also, guide those who are interested in continuing our work.

1 Introducción

1.1 Motivación

Los mercados son el motor económico de nuestra sociedad. Casi cualquier transacción se realiza en un mercado o en un entorno que se puede asemejar a un mercado. Encontrar una buena estrategia de venta es esencial para conseguir beneficios.

La mayoría de los mercados son entornos de información asimétrica como los descritos por el economista George Akerlof en su ensayo de 1970 "The Market for Lemons: Quality Uncertainty and the Market Mechanism"[1]. Como describe Akerlof, en un mercado generalmente el vendedor conoce mejor el producto a vender que el comprador. Esto le permite dar una información sesgada respecto a la calidad del producto y llevar a engaño al comprador. En nuestra vida diaria estamos sumergidos en ejemplos de este comportamiento, y prueba de ello es el hecho de que la publicidad es una gran industria y que hemos aprendido a convivir con ella y a no dejarnos seducir por cada oferta que nos hacen.

Un comprador, naturalmente, quiere asegurar que el producto en el que va a invertir su dinero le satisfará y, conocedor de que el vendedor intentará venderle el producto a toda costa, tiene una natural desconfianza ante una transacción. El comprador debe confiar en el vendedor para que la transacción se lleve a cabo.

El ensayo de Akerlof describe lo que el llama “Market for Lemons”. En este mercado los compradores no pueden conocer antes de comprar un producto su calidad exacta, es lo que Akerlof denomina un mercado donde la información es asimétrica. El ejemplo que pone es el del mercado de los coches usados. Dado que quien compra no conoce suficientemente el estado en el que está el coche por el que va a pagar, no estará dispuestos a pagar más que lo que vale un coche de calidad media. Los vendedores a sabiendas de este comportamiento de los compradores no querrán poner a la venta en ese mercado coches cuya calidad esté por encima de la media, dado que los compradores no están dispuestos a pagar su precio. Al retirar los productos buenos del mercado, se baja la calidad media esperada y, al bajar ésta, baja también el precio máximo que pagará un comprador. Este círculo vicioso llevará a un colapso del mercado.

En la realidad esto no ocurre, porque los vendedores consiguen que los compradores depositen su confianza en ellos y los convencen de que el producto que van a comprar realmente vale lo que van a pagar por él. Un comprador puede depositar su confianza en un vendedor porque le parece honesto, porque tiene una gran reputación, porque haya tenido buenas experiencias previas con él o simplemente porque no considera la inversión lo suficientemente costosa para dedicarle tiempo a buscar otros vendedores.

En cualquier caso, es evidente que la confianza a la hora de realizar una compra y, a más alto nivel, para que funciones un mercado, es esencial. Y para un vendedor, asumiendo que el fin último de un vendedor es ganar el máximo dinero posible, es muy importante labrarse una buena reputación y comportarse con los clientes de manera que estos confíen en él para que hagan sus compras a través de él.

El problema estriba en que un vendedor no sabe como ha de comportarse para maximizar su beneficio ganándose la confianza de los clientes y mejorando su reputación. Obtener una estrategia buena requiere realizar experimentos en su comportamiento con los clientes que podrían ser costosos para él, tanto en tiempo como en dinero.

Una herramienta que permita analizar como influyen ciertos factores y comportamientos en la reputación y en la confianza que un cliente tiene en un vendedor es, por tanto, un valioso aliado para cualquiera que quiera obtener nuevas estrategias de venta.

Utilizar la informática para realizar simulaciones sobre la confianza y la reputación en distintos entornos es algo que se viene haciendo desde hace algún tiempo. Existen entornos de simulación como el Art Test-bed[23] o el SuppWorld framework[4] que permiten simular la evolución de la confianza y la reputación en distintas situaciones. En la tesis de Jordi Sabater[4] se listan varios simuladores de este campo.

La mayoría de estos trabajos tienen un enfoque teórico, más encaminado a obtener datos sobre como se comportan la confianza y la reputación en transacciones generalizadas. Estas conclusiones no se pueden extrapolar directamente a un mercado, que es un entorno complejo donde influyen muchas variables.

El propósito de este proyecto es intentar desarrollar un entorno de simulación que rellene ese vacío.

1.2 Objetivos

El objetivo principal es proporcionar una plataforma para el estudio de la confianza y la reputación en un mercado. Para satisfacer este objetivo se utilizará la simulación con agentes.

Hemos planteado como objetivos crear un sistema multi-agente para el estudio de la confianza y la reputación con las siguientes características:

- Se intentará modelizar para la simulación una cadena de mercados que se asemeje lo más posible en su comportamiento y funcionamiento a la de una situación real, pero sin desviarse hacia detalles que no influyan o entorpezcan el estudio de la confianza y la reputación. Se ha tomado como objeto de estudio los MERcados Centrales de Abastecimiento de Madrid, MercaMadrid [2], para que sirvan de modelo en lo referente a funcionamiento y estructura de la cadena de mercados.

- Asimismo deberá ser extensible y moldeable para poder simular otro tipo de entornos.

- El sistema permitirá realizar simulaciones con diferentes parámetros y observar la evolución de los agentes en función de estos parámetros.

- Asimismo el diseño debe permitir que la estructura de mercados se pueda variar, permitiendo así el desarrollo de nuevos entornos de simulación con estructura diferente a la de Mercamadrid.

- Se proporcionará documentación que permita la futura extensión del sistema a otros desarrolladores.

- Tal como se hace en Mercamadrid, se implementará además un sistema de reglas o leyes mediante el cual un agente denominado como Juez dispondrá de la autoridad de resolver conflictos y penalizar a los agentes. Esto contribuirá a mejorar el comportamiento del sistema.

2 Estructura General

2.1 Documentación

2.1.1 Confianza y Reputación

La informática hace ya tiempo que pasó de una máquina aislada a ser a una red de sistemas o sistemas distribuidos y la inteligencia artificial está sufriendo el mismo proceso de cambio, pasando de una inteligencia aislada a una inteligencia colectiva y social. Por ello, la confianza y la reputación están ganando relevancia y un gran terreno en el mundo de la informática que cada vez presta más atención a los sistemas multi-agente y sociedades virtuales inteligentes.

Como explica Jordi Sabater en su tesis "Trust and reputation for agent societies" [4], La confianza es necesaria en nuestro día a día, es la razón por la cual la sociedad se mantiene unida. Sin la confianza, los gobiernos no podrían gobernar y mantener un equilibrio en las sociedades, asimismo, las personas no podrían cooperar los unos con los otros. La confianza nos ayuda a reducir la complejidad de las decisiones humanas que se ven afectadas por innumerables factores.

De forma similar, la reputación es un concepto universal conocido desde las antiguas civilizaciones de este mundo. La reputación juega un importante papel en la organización de nuestras sociedades y es un elemento clave en el que nos basamos para generar una confianza en otra persona.

Por ejemplo, en las comunidades online se utilizan mecanismos de reputación que permiten describir la calidad de los servicios de cada agente virtual. Un estudio demostró que una buena reputación puede conseguir incrementar en un 7.6% el dinero que recibes al prestar un servicio.

El concepto de confianza y reputación aplicado a comunidades virtuales de agentes es una nueva disciplina utilizada para incluir estos dos mecanismos de control humanos en sistemas mejorando su fiabilidad y progreso.

En primer lugar, denominaremos "confianza" como la fiabilidad que un agente estima sobre otro. Es tarea del agente almacenar y actualizar la confianza, así como tener en cuenta la confianza a la hora de interactuar.

Por otra parte, usaremos el término "reputación" para determinar la fiabilidad que un agente posee en el colectivo del mercado de una forma más global.

El modo de saber qué confianza depositar en un agente, si no se dispone de información previa relativa a él, normalmente involucra preguntarle a otro que sí ha tenido experiencias previas con él qué opinión le merece.

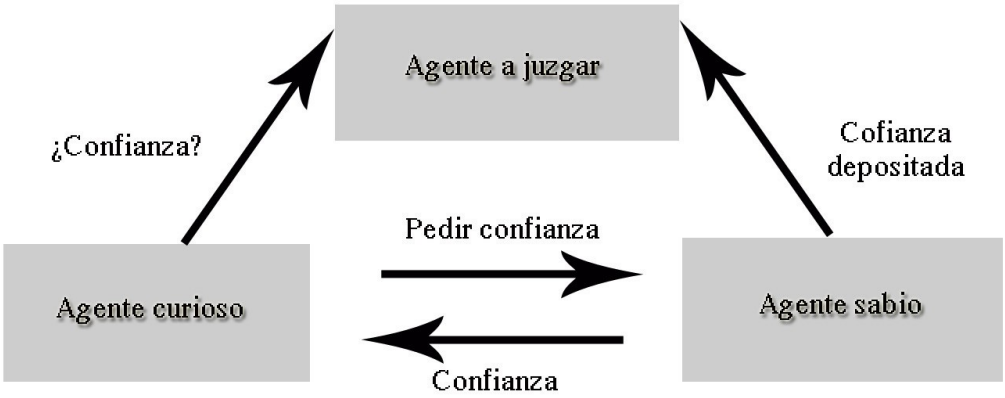


Figura 2.1.1.1 - Obtención de la confianza.

Según se relata en el trabajo sobre ART, Testbed competition de Iván García-Magariño [23], el objetivo que se persigue con el modelo de confianza y reputación es disminuir al máximo los efectos negativos de los agentes poco fiables. Un sistema robusto debería excluir a dichos agentes del sistema. Para ello, cada agente intenta predecir la fiabilidad de los agentes que le rodean. Ya sea por experiencias anteriores con dicho agente (confianza) o por la reputación que el mercado le tiene asignado. Los detalles sobre el modelo de la confianza y la reputación se encuentran en el apartado *Detalles de Implementación*.

2.1.2 *Simulación con agentes*

En este apartado se va a explicar brevemente en qué consiste la simulación con agentes, para ello se explicarán los conceptos de agente y sistema multi-agente.

● *¿Qué es un Agente?*

El concepto de "agente" caracteriza a una entidad software con una arquitectura robusta y adaptable que puede funcionar en distintos entornos o plataformas computacionales y es capaz de realizar de forma "inteligente" y autónoma distintos objetivos intercambiando información con el entorno o con otros agentes humanos o computacionales. *(Definición obtenida de "Agentes software y sistemas multi-agente: conceptos, arquitecturas y aplicaciones" [5])*

Las características más destacables del comportamiento de un Agente son:

- Funcionamiento continuo y autónomo.
- Comunicación con el entorno y con otros agentes (posiblemente humanos) por medio de un lenguaje o formalismo de comunicación (protocolos).
- Robustez.
- Adaptabilidad como capacidad de realizar objetivos y tareas en distintos dominios de forma incremental y flexible. La adaptación implica ser consciente del entorno y reconfigurarse como respuesta a él. Esto puede ser alcanzado mediante la elección de reglas de resolución de problemas alternativas o algoritmos, o mediante el descubrimiento de estrategias de resolución de problemas.

También existen otras capacidades importantes de un agente inteligente que podemos encontramos, aunque no necesariamente. Estas son:

- Movilidad

  Los agentes móviles son capaces de desplazarse entre los nodos de una red y ejecutarse en distintas plataformas.

- Capacidad de razonamiento y aprendizaje

Ambos aspectos son necesarios para que el agente sea capaz de comportarse inteligentemente. El aprendizaje puede implementarse mediante la técnica de prueba y error, en cuyo caso implicaría una capacidad de análisis de comportamiento y éxito. De forma alternativa, el aprendizaje puede proceder mediante ejemplos y generalización, lo que requeriría una capacidad de abstracción y generalización.

Como bien se explica en "Software Agents" [6], Nwana propone una tipología de agentes que identifica diferentes dimensiones de clasificación según muestra el esquema siguiente:

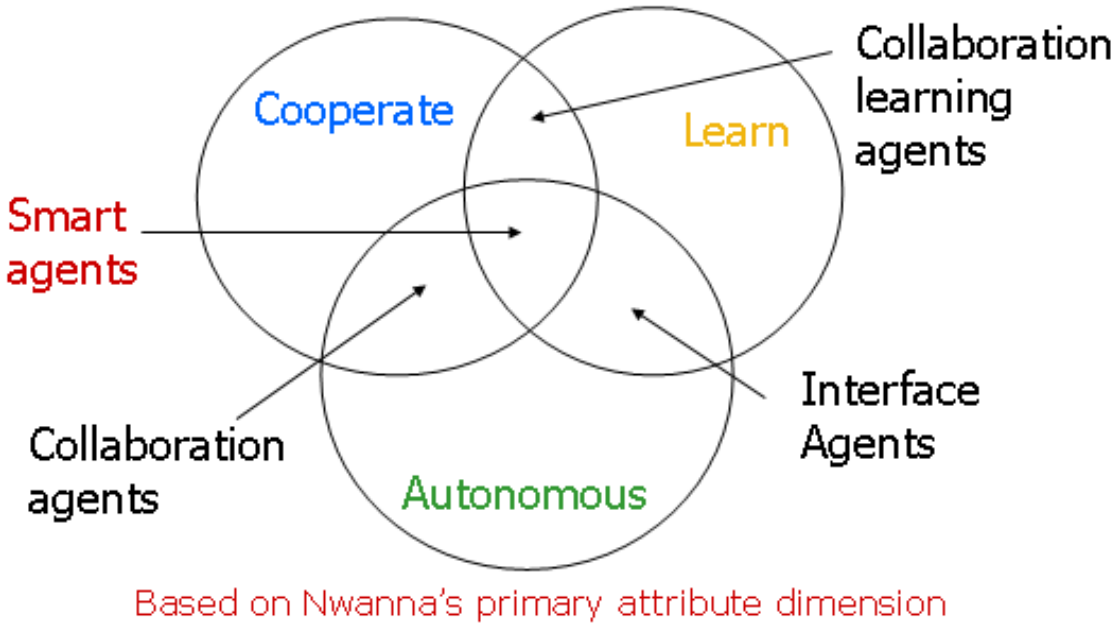


Figura 2.1.2.1 - Tipos de agentes.

Según esta tipología, nos encontramos con las siguientes categorías de agentes: agentes colaboradores, agentes interfaz, agentes móviles, agentes de información/Internet, agentes reactivos, agentes híbridos y agentes inteligentes.

2.1.3 *Sistemas multi-agente*

Un sistema multi-agente se trata de un sistema compuesto por varios agentes, que son capaces de alcanzar objetivos colectivamente que serían difíciles para un solo agente o para sistemas monolíticos. Los sistemas multi-agente pueden manifestar auto-organización y comportamientos complejos

incluso cuando las estrategias individuales de todos sus agentes son simples. Los agentes operan y existen en este entorno que típicamente es tanto computacional como físico. Entorno que proveerá una infraestructura para las interacciones entre agentes y la infraestructura incluirá protocolos para que los agentes se comuniquen y protocolos para que interaccionen *(Definición obtenida de "Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence" [7]).*

Las características deseables de los sistemas multi-agente son:

- Los entornos multi-agente proveen una infraestructura especificando los protocolos de interacción y comunicación.
- Los entornos multi-agente son, normalmente, abiertos y no tienen un diseño centralizado. Esto es, permitir de manera pública a cualquier desarrollador contribuir con uno o varios agentes
- Los sistemas multi-agente contienen agentes que son autónomos y distribuidos, y podrían trabajar según el interés propio o colaborando con otros.

Los sistemas inteligentes no funcionan de forma aislada, es decir, suelen ser una parte pequeña del entorno al que pertenecen, y el entorno típicamente contiene otros sistemas inteligentes. De esta forma, parece razonable ver a dichos sistemas desde una perspectiva social.

Un grupo de agentes puede formar una pequeña sociedad en la que cada uno juega un rol distinto. El grupo define los roles, y los roles definen los compromisos. Cuando un agente se une a un grupo adquiere automáticamente los compromisos asociados al mismo. De esta forma se define el contexto social en el que los agentes interaccionan.

En nuestro caso, hemos creado un sistema multi-agente basado en la confianza y la reputación.

Para un buen funcionamiento de este tipo de sistema multi-agente, son deseables las siguientes cualidades:

- Precisión en la predicción: Se pretende predecir correctamente lo fiable que va a ser un agente en la siguiente transmisión de información. Para ello se utilizan la confianza y la reputación.

- Adaptativos: Se pretende que el sistema sea robusto frente a los cambios de conducta de agentes individuales. Es decir, prever que un agente fiable, de repente, empiece a desarrollar una conducta maliciosa. Esto requiere revisar o actualizar constantemente las confianzas.

- Exclusión de los Agentes No Fiables: Se pretende que los agentes excluyan a los agentes no fiables, mediante no interaccionar con ellos.

- Interacciones Adecuada entre Agentes: Cada agente del sistema debe elegir con qué agentes interaccionar para recibir o dar información, de la manera más favorable.

En resumen, un sistema multi-agente nos provee de un entorno computacional que es necesario para la interacción de varios agentes software. Los elementos del sistema capacitan al agente para comunicarse, cooperar y negociar mientras actúan en el interés propio o de su sociedad.

2.2 - *Caso de estudio*

2.2.1 Descripción de Mercamadrid.

Mercamadrid es un centro de negocios de la alimentación, un polígono alimentario situado en Madrid con proyección nacional e internacional, que abastece a más de 9 millones de habitantes. Mercamadrid reúne tanto a los Mercados Centrales de Pescados y Frutas y Hortalizas de Madrid, como al Mercado de Carnes, a empresas polivalentes especializadas en el sector alimentario, y a un amplio rango de empresas de servicios: frío industrial, conservación, logística, transportes, manipulación.

Mercamadrid funciona como un mercado mayorista. Los clientes acuden al mercado mayorista a comprar productos en los que están interesados. Allí encuentran a varios mayoristas con el producto que buscan, y le piden condiciones y precio por una cantidad de dicho producto. El mayorista les contesta si puede conseguir dicha cantidad y expone las condiciones de venta de dicho producto. Si al comprador le interesa acepta, si no, lo rechaza posiblemente proponiendo otras condiciones alternativas. Si se llega a un acuerdo el mayorista busca en los exportadores que tienen el producto referido y sus condiciones de venta. Una vez visto cual le interesa más, cierra el trato con ese exportador. A partir de aquí, el exportador se dirige a los que obtienen los productos y participa en las subastas para hacerse con el producto que quiere. Cuando consigue hacerse con el producto, se lo entrega al mayorista, y éste al cliente.

Como toda empresa, Mercamadrid posee un reglamento amplio y detallado y un organismo que se encarga de que éste se haga cumplir, con sus consecuentes multas o penalizaciones para aquellos que no se rijan por sus normas. Este apartado es importante para nuestra simulación, tal como se explica en el apartado de adaptación. Se tratarán en ese apartado las normativas hemos escogido como más importantes para nuestro sistema.

2.3 Adaptación.

2.3.1 Visión general.

Como se menciona en los objetivos, el sistema constituye un entorno para el estudio de la evolución de la confianza y la reputación en respuesta a diferentes comportamientos. Para dar mayor realismo al entorno y permitir un estudio en un ambiente que se asemeje a la realidad, se ha intentado asemejar el funcionamiento del sistema al de mercados reales para lo cual se ha llevado a cabo una importante labor de documentación en las fuentes que aparecen detalladas al final de esta memoria.

Sin embargo, no es el objetivo de este sistema modelizar y simular al mínimo detalle una cadena de mercados real. Por ello, no se toman en cuenta aspectos que no tienen influencia en el proceso de compra como tal, que es el centro de la simulación puesto que es en ese momento donde se generan y se utilizan las confianzas y reputaciones. Entre estos aspectos que no han sido tomados en cuenta se encuentran los siguientes:

- Las cuestiones de escala no han sido tomadas en cuenta puesto que no son relevantes para obtener conclusiones sobre el comportamiento de los agentes y sobre el manejo de la confianza y la reputación, fin último de este simulador. Una estrategia de venta o una forma de computar la reputación de un agente se puede simular sin tener en cuenta si las unidades que maneja representan un solo producto o grandes lotes de ellos. Este nivel de abstracción facilita la simulación, ya que sería imposible simular una cadena de mercados compleja como la de este sistema si se tuviesen en cuenta todo lo que ocurre en ella, puesto que los costes computacionales se dispararían.

- Ha sido ignorado también lo referente a los horarios de venta y calendarios de los mercados por no aportar nada a la simulación. Se puede entender que el sistema solo simula los momentos en los que el mercado está activo, ignorando los horarios de cierre y días festivos.

- Tampoco se simula el apartado de transporte de las mercancías de un mercado a otro ni dentro del mismo. Se supone que ese proceso corre a cargo del agente que quiera llevar el producto de un mercado a otro y por tanto sale del ámbito que nos interesa.

- Se han obviado por la misma razón tasas, impuestos y normativas que, si bien atañen a los vendedores en los mercados reales, no son aplicables en este sistema. Dentro de esta categoría se encuentran la gran mayoría de las normas incluidas en los diferentes reglamentos de MercaMadrid [8,9,10] que se refieren a mantenimiento del local, uso de cámaras frigoríficas y demás que aquí no han sido tenidas en cuenta.

- Por otra parte, MercaMadrid se subdivide en tres mercados diferentes, el Mercado Central de Carnes, el Mercado Central de Pescados y el Mercado Central de Frutas y Hortalizas (además de diferentes zonas de servicios y demás), que tienen distintas normativas en lo referente a utilización de sus instalaciones [8,9,10] pero cuyo funcionamiento es básicamente el mismo. Por ello, no hemos prestado especial atención al tipo de productos que se venden en el sistema y hemos cogido las características generales los tres mercados para modelar los nuestros. No obstante, el diseño está abierto a poner normativas estrictas y específicas en los mercados que pongamos en la cadena, pudiéndose hacer las diferenciaciones que se consideren apropiadas.

2.3.2 La cadena de mercados.

La cadena de producción en la que se encuentra MercaMadrid es muy compleja. Un vistazo a su página web [2] nos da una idea de sus dimensiones y nos permite adivinar que los productos que llegan a los diferentes mercados de MercaMadrid siguen caminos muy diversos y los mayoristas que venden allí son, también, de naturaleza muy diversa. En efecto, una investigación más exhaustiva nos permite corroborar que dentro de la gran diversidad de empresas que allí actúan hay algunas que producen directamente los productos que venden, otras que los compran a los productores y otras que los compran a intermediarios.

Por tanto, la topología de los mercados involucrados en el camino de un producto desde su centro de producción hasta el consumidor de MercaMadrid es muy compleja e inabarcable.

A la hora de diseñar el sistema hemos en cuenta la variedad de topologías que puede tener una cadena de producción, y hemos intentado proporcionar la máxima flexibilidad posible a la hora de construir una aplicación que simule una cadena de mercados. De esta manera, el diseño permite que un agente actúe en tantos mercados como quiera, que haya tantos mercados como se necesite y que estén conectados de cualquier manera.

La cadena que hemos implementado nosotros tiene una topología lineal, es decir, cada agente compra en un mercado y vende en otro. Además, todos los agentes que compran en un mercado comparten además el mercado en el que venden. Esto es así no por obligación del diseño, ya que el diseño permite topologías mucho más complejas, sino para obtener resultados comparables a los obtenidos con otros sistemas con menos variabilidad.

Figura 2.3.2.1 – Cadena de Mercados.

En este diagrama se aprecia que los productos generan los productores en sus granjas, estos productores los venden en un mercado que opera mediante subasta a unos intermediarios que a su vez se los venden en otro mercado (que opera según el protocolo contract-net [11]) a los vendedores de MercaMadrid que finalmente los venden allí a los compradores[1].

A continuación detallamos cada uno de los elementos de la cadena y la adaptación que ha sido necesaria para su modelización.

1 Los compradores de MercaMadrid son a su vez mayoristas.

2.3.3 *Mercados.*

Son los lugares en los que tiene lugar una transacción. En el mundo real muchas de estas transacciones no se llevan a cabo en un lugar físico sino mediante contacto directo entre las partes interesadas, aunque la mayoría de las veces sí se producen en lugares destinados a tal efecto que siguen una serie de normas.

Los mercados de nuestro sistema representan cualquiera de esas dos opciones o, más generalmente, cualquier ámbito en el que varios agentes realizan transacciones.

En este sistema, un mercado es un medio para que uno o varios vendedores realicen transacciones con uno o varios compradores.

Los mercados proporcionan el canal mediante el cual se aplican uno o varios protocolos que son los que permiten las transacciones, de manera que, para operar en un mercado, tanto compradores como vendedores deben conocer al menos uno de los protocolos que se utilizan en él.

Son los protocolos que proporciona el mercado los que dictaminan el tipo de bienes que se intercambian en ellos, ya sean dinero, productos, información...

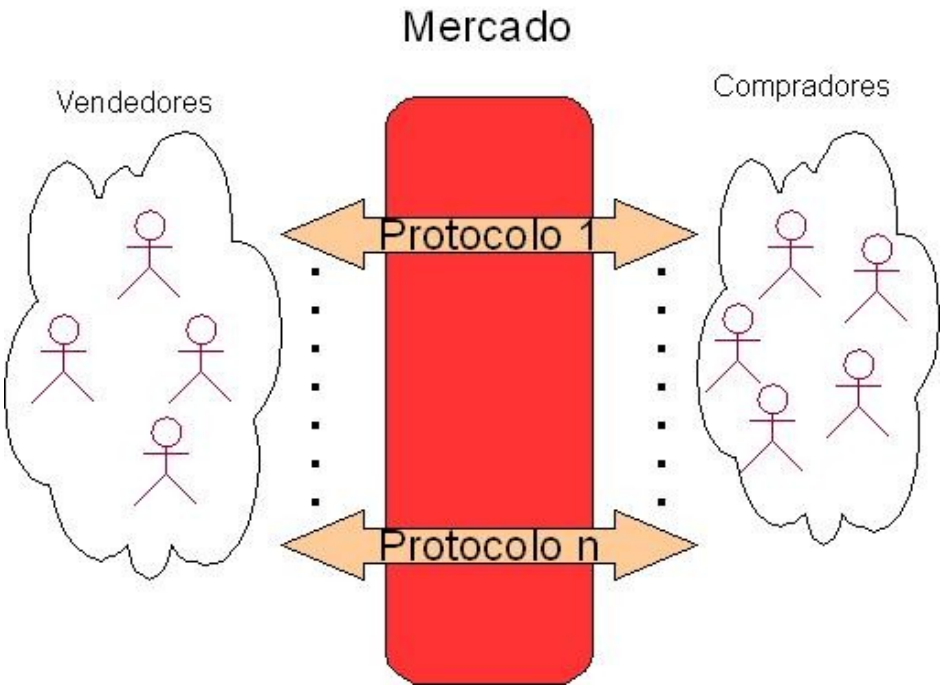


Figura 2.3.3.1 – Visión general de un mercado.

Los mercados llevan, además, un registro de todas las transacciones que se realizan en ellos. Este registro sirve para poder observar los resultados que obtienen los agentes y para tener datos sobre los que poder aplicar las normas del mercado si este las tuviese. El registro se corresponde con la documentación administrativa y con los distintos libros de cuentas[1] que deben llevar los vendedores de MercaMadrid y que se menciona en los reglamentos de los diferentes mercados [8,9,10].

Si el mercado tiene un reglamento puede tener un juez que vele porque éste se cumpla utilizando para ello los datos almacenados en el registro.

1 Los reglamentos de los mercados de MercaMadrid indican que cada vendedor debe llevar un libro de cuentas en el que aparezcan reflejadas todas sus transacciones y que deberá ser consultado por personal del mercado siempre que se solicite. Por razones de diseño (simplificar el acceso a los resultados), hemos descargado de esa responsabilidad al vendedor y el registro se rellena automáticamente con cada transacción, si bien el vendedor puede consultar su sección del registro siempre que quiera y es libre de llevar uno propio.
Por otra parte, el vendedor puede hacer una venta que no conste en el registro contactando directamente con el comprador, pero será penalizado por ello si el mercado tiene normas que lo prohíban y medios para percatarse (generalmente un juez).

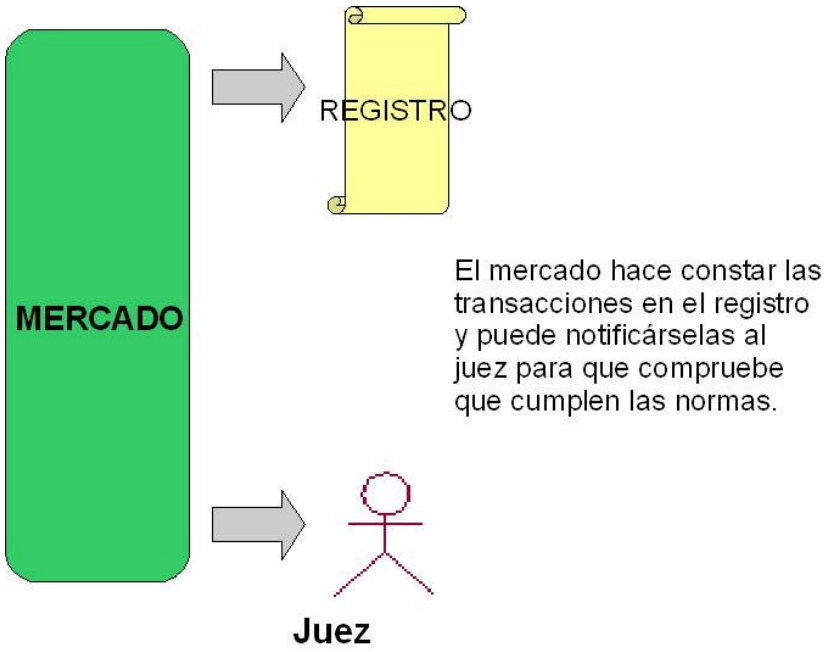


Figura 2.3.3.2 – Mercado con juez y registro.

Otra simplificación llevada a cabo consiste en que los mercados de nuestro sistema son estáticos a lo largo de una simulación. En mercados reales un vendedor puede dejar su puesto vacante o a otro vendedor que lo reemplace si así lo desea. Sin embargo, en nuestro sistema, más enfocado a una simulación estática como la mayoría de entornos de simulación de confianza y reputación, esto no se permite. La complejidad del sistema aumentaría demasiado como para obtener resultados comparables a la realidad. No obstante, esta posibilidad está contemplada en el apartado de trabajos futuros y se ha procurado hacer un diseño que lo permitiese con pocas modificaciones.

2.3.4 *Protocolos.*

Los protocolos son los mecanismos de entendimiento entre agentes para que interactúen entre ellos. Solo especifican un conjunto de operaciones, por lo que estos protocolos necesitarán de un mercado que sirva de soporte para su aplicación.

En la realidad los protocolos de compraventa pueden estar especificados en la normativa de los mercados, en el caso de protocolos complejos como los de lonjas o subastas, o dejarse sin especificar. De esta manera, se terminan utilizando protocolos más o menos variables pero de sobra conocidos por cualquiera que ha hecho alguna compra en su vida.

En nuestro sistema se pueden utilizar tantos protocolos como se deseen especificar. Para la cadena de mercados que hemos implementado de ejemplo hemos implementado dos protocolos, el protocolo contract-net y el protocolo subasta (inglesa), por considerarlos representativos y diferentes entre sí.

- ContractNet. El protocolo contract-net es de sobra conocido por cualquiera, pues se refiere al protocolo que se utiliza en cualquier transacción normal del día a día. El nombre contract-net corresponde a una especificación formal de este protocolo realizada por la Foundation for Inteligent Physical Agents [11,12].

  - Nuestra implementación del protocolo se puede representar con este diagrama.

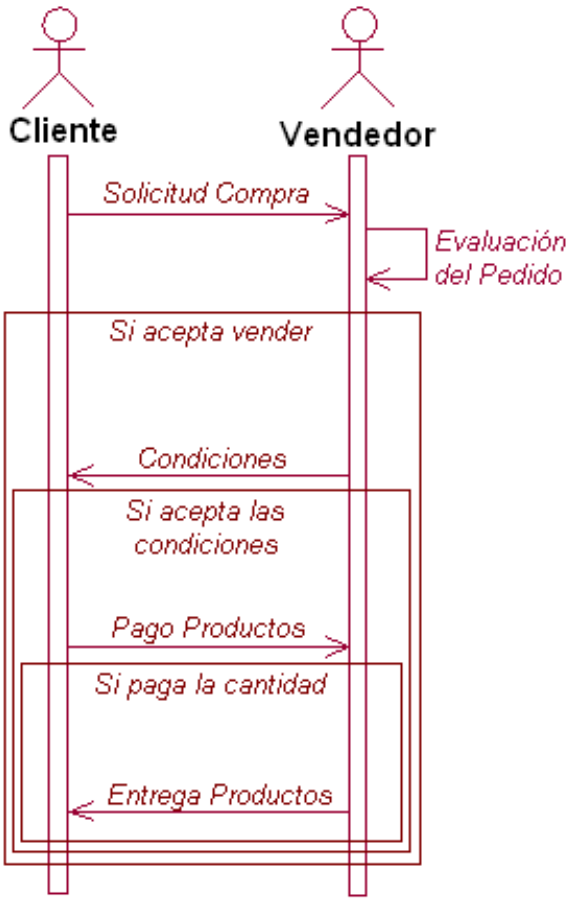


Figura 2.3.4.1 – Comunicación en el protocolo contract-net.

- En el diagrama podemos observar que la transacción la inicia el comprador cuando solicita un producto al vendedor. Éste analiza la oferta y, si le interesa, contesta al comprador exponiendo sus condiciones de venta. Transcurrido un tiempo, si el comprador decide que quiere aceptar la oferta, el pago de la misma informa al vendedor de que la oferta ha sido aceptada, por lo que solo restará que, si está satisfecho, entregue los productos al comprador.

- El protocolo es fiel al especificado por la FIPA, si bien algunos de los mensajes se han dejado implícitos en nuestra implementación para no sobrecargar la simulación. Comparando el diagrama de la FIPA y el nuestro se observa que los mensajes implícitos son los que se refieren a rechazo de la oferta o a rechazo de las condiciones. Al no recibir una respuesta afirmativa al cabo de un tiempo, el agente asumirá que su propuesta ha sido rechazada, evitando así un mensaje innecesario.

- Subasta inglesa. La subasta inglesa es el protocolo de compra-venta que estamos acostumbrados a ver en lo que coloquialmente conocemos sólo como subastas. Son ejemplos de este tipo de compra-venta las subastas de arte, las pujas públicas, etc.

  - En una subasta inglesa, inicialmente el vendedor fija un precio bajo que los compradores irán subiendo hasta que se alcanza una puja que nadie está dispuesto a rebasar[20,21].

  - Este diagrama ilustra nuestra implementación del protocolo subasta inglesa.

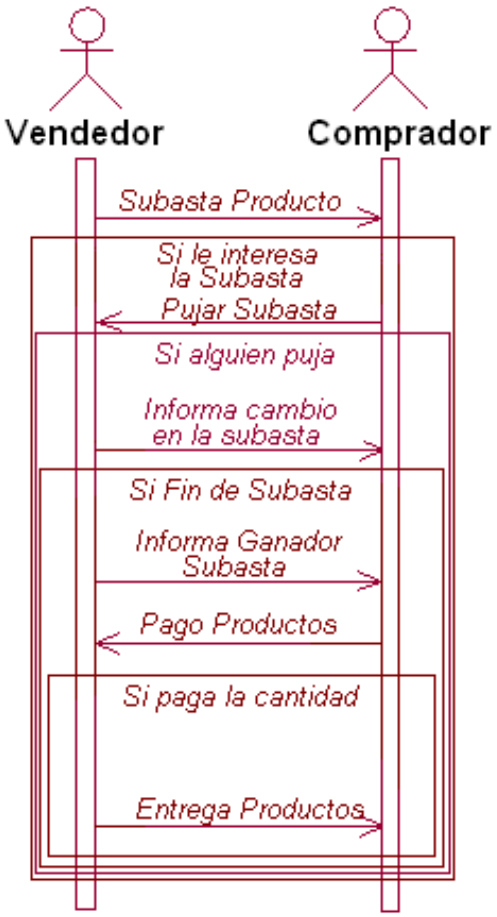


Figura 2.3.4.2 – Comunicación en el protocolo subasta.

- En esta ocasión es el vendedor el iniciador, pues notifica a los compradores que está comenzando una nueva subasta. El comprador que esté interesado hará su puja. Por cada cambio en el valor máximo de la puja los compradores reciben una notificación, volviendo a pujar si así lo desean. Llegado un tiempo elegido por el vendedor la subasta termina, se informa al máximo pujador y al recibir el pago se entregan los productos.

- Protocolo de intercambio de reputación/confianza.

  - Es el protocolo que utilizan los compradores para intercambiarse su opinión respecto a un vendedor.

- Es una adaptación del protocolo contract net en la que el objeto de la venta es información.

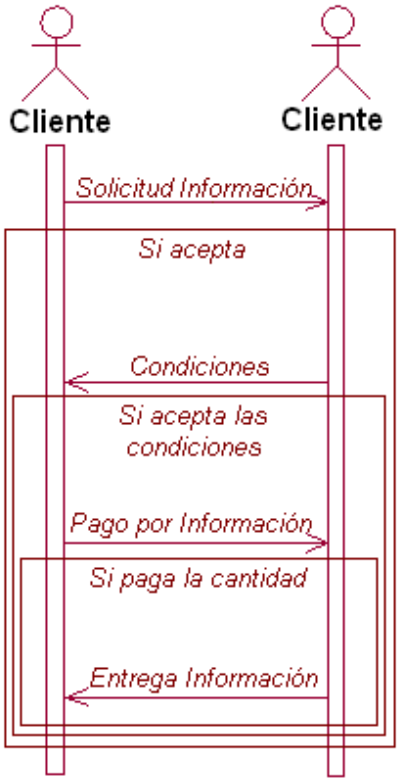


Figura 2.3.4.3 – Comunicación en el protocolo de intercambio de confianza.

- También se pueden prefijar las condiciones para este tipo de transacciones y realizar solamente los últimos dos pasos.
- Este protocolo también describe como un comprador puede solicitar la reputación de un vendedor al mercado. Para hacerlo, deberá pagar el precio establecido si hubiese alguno, y el mercado directamente le informará de la reputación del vendedor en cuestión.

- Protocolo de reclamaciones.
  - Este protocolo describe lo que debe hacer un consumidor para presentar una queja al mercado.

- En la nuestra implementación consiste solamente en una especificación de los datos que debe contener la queja (reclamante, reclamado, productos a los que afecta, motivo y, si procede, cantidad acordada).

2.3.5 *Juez.*

El juez es un agente del que opcionalmente pueden disponer los mercados para que vele por el cumplimiento se sus normativas si éstas existieran y para que aplique las sanciones que sean necesarias.

En MercaMadrid, el mercado que hemos tomado de modelo a la hora de especificar las funciones del juez, los reglamentos se encargan de definir cuáles son las funciones del juez y quién ejerce ese papel.[1]

Es papel del juez, además, atender a las reclamaciones de los usuarios y valorarlas para tomar medidas si fuese necesario.

Las normativas de MercaMadrid son muy concretas respecto a materias como las condiciones sanitarias, el transporte de productos, la utilización de neveras... Muchas de esas normas no son aplicables en nuestra simulación, por lo tanto han sido ignoradas. Han sido tenidas en cuenta las normas referentes a las transacciones.

En base a esto, hemos modelado un juez con las siguientes funciones:

- Atender a las reclamaciones de los consumidores.
- Velar por el cumplimiento de las reglas que se le indiquen.

El juez que hemos implementado en la cadena de mercados vela por que los vendedores no gasten más dinero del que han ganado vendiendo sumado al que tenían inicialmente, puesto que esto demostraría que han ganado dinero con

1 El reglamento del Mercado Central de Carnes dice que la Administración del Mercado corresponde al Gerente del Matadero Municipal y especifica sus funciones, entre las que se encuentran recibir las reclamaciones de los usuarios y aplicar las normas que se indican en ese mismo documento. En el Mercado Central de Pescados y en el Mercado Central de Frutas y Hortalizas esa responsabilidad recae en un Jefe de Mercado, nombrado por la dirección de MercaMadrid, que aplica las normas que aparecen en el reglamento y que son dictadas por la Empresa Mixta.

ventas que no han sido declaradas al mercado.[1] Vigilará además que un vendedor que ha sido sancionado cumpla su pena.

El juez también atiende a las quejas de los compradores, las valora y si se demuestra que el vendedor ha vendido productos de calidad o en cantidad inferior a la acordada[2], lo sancionará.

1 Reglamento de Funcionamiento del Mercado Central de Pescados - artículo 115 apartados d,e y f.
Reglamento de Funcionamiento del Mercado Central de Carnes - artículo 56 apartados g y h.
Reglamento de Funcionamiento del Mercado Central de Frutas y Hortalizas - artículo 104 apartados d,e y f.

2 Reglamento de Funcionamiento del Mercado Central de Pescados - artículo 115 apartado c.
Reglamento de Funcionamiento del Mercado Central de Carnes - artículo 56 apartado e.
Reglamento de Funcionamiento del Mercado Central de Frutas y Hortalizas - artículo 104 apartado c.

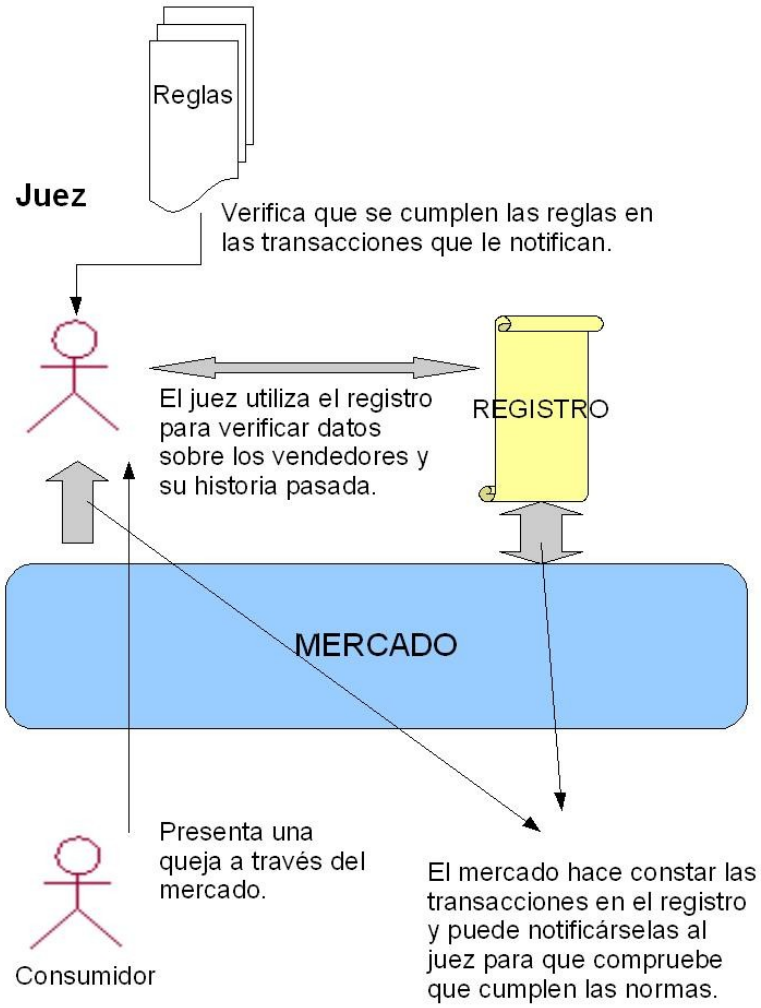


Figura 2.3.5.1 – Interactividad del juez.

Las infracciones que detecte el juez se dividen en leves o graves según la cuantía defraudada y las sanciones aplicadas dependerán del tipo de la infracción. Estas sanciones han sido elegidas basándonos en las que se aplican en los mercados de MercaMadrid que aparecen detalladas en los reglamentos. La reiteración de faltas leves se considerará una falta grave[1].

1 Reglamento de Funcionamiento del Mercado Central de Pescados - artículo 114 y 115 apartado a. Reglamento de Funcionamiento del Mercado Central de Carnes - artículo 55 y 56 apartado a. Reglamento de Funcionamiento del Mercado Central de Frutas y Hortalizas - artículo 103 y 104 apartado a.

- Las infracciones leves se sancionarán con prohibición de venta durante un período proporcional a la cantidad defraudada así como con una pérdida de reputación.

- Las infracciones graves se sancionarán con una prohibición de venta durante un período proporcional a la cantidad defraudada pero con un factor de proporcionalidad mayor, con una pérdida de reputación mayor y además con una sanción económica también proporcional a la cantidad defraudada.

Además se pueden añadir reglas respecto a los productos vendidos o a la forma de las transacciones para que el juez compruebe que se cumplen en toda compraventa.

Puesto que el fin principal del sistema es simular la evolución de la confianza y la reputación, hemos hecho que las sanciones del juez afecten negativamente a la reputación de los agentes. Obviamente en el mundo real no está especificado que al sancionar a un agente su reputación baje, porque la reputación es una cualidad abstracta que no depende de nadie en concreto, pero es evidente que un vendedor que sea muy sancionado tendrá peor reputación que otro al que no se sanciona nunca. La forma en la que el juez baja la reputación, o si la baja siquiera, se debe describir al diseñar al juez.

Por otra parte, el diseño no limita la complejidad del juez ni las funciones que éste cumple, pudiendo utilizarse jueces con funcionalidades complejas que atiendan a uno o varios mercados y verifiquen normas complejas.

2.3.6 *Productos.*

Son los objetos de la compraventa. Poseen cualidades que les dan un valor. En nuestra implementación la única cualidad que poseen es la antigüedad y de ahí se deriva su calidad. Se pueden añadir otras características complejas que se modifiquen a lo largo del tiempo.

En el ejemplo hemos puesto cuatro productos diferentes (pollos, cerdos, corderos y terneros) que sólo se diferencian en el nombre, aunque en las simulaciones solamente hemos utilizado uno de ellos para no introducir demasiadas variables.

El sistema está diseñado de manera que se puedan añadir productos con funcionalidades propias y cuya calidad dependa de varios factores.

2.3.7 *Centros de producción.*

Por centros de producción entendemos cualquier entidad que fabrica los productos de los que nos ocupamos.

En la realidad estos centros de producción pueden ser granjas o criaderos (para los mercados de carne), bancos de pesca o piscifactorías (para los mercados de pescado) o huertos u otras formas de cultivo (para los mercados de frutas y hortalizas).

El funcionamiento de los centros de producción varía mucho de unos a otros. Los mecanismos y vicisitudes de los centros de producción no son relevantes para la simulación de la confianza y la reputación. Por ello, en nuestro sistema los centros de producción son entidades que transforman dinero en nuevos productos al cabo de un tiempo.

Las granjas retiran dinero de la cadena e introducen nuevos productos

El mecanismo de operación de los centros de producción en nuestro sistema consiste en que reciben un pago para fabricar un producto y, al cabo de un tiempo de producción que depende del tipo de producto, se entrega la cantidad solicitada de productos de ese tipo.

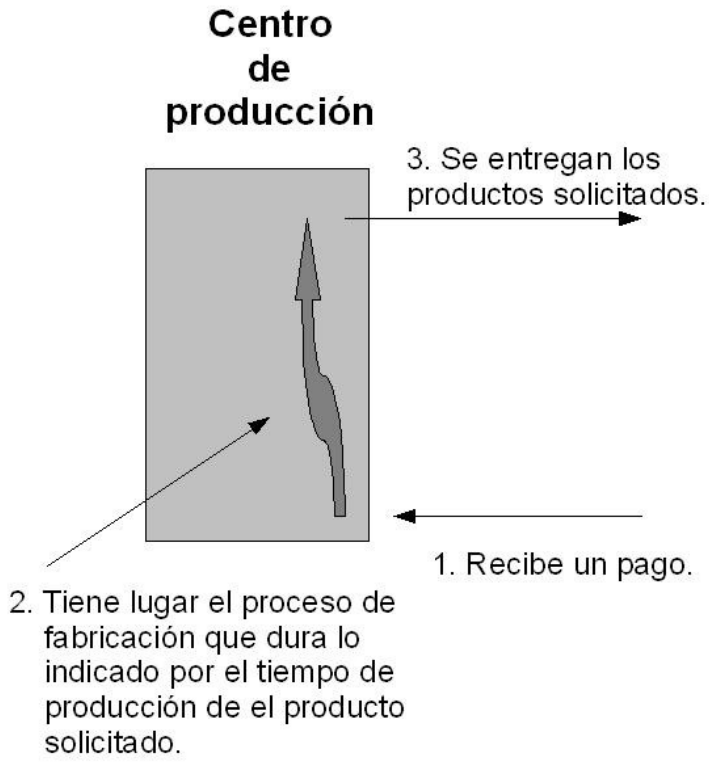


Figura 2.3.7.1 – Centro de producción.

Los centros de producción introducen variabilidad en la cadena de mercados, ya que el tiempo de producción no tiene por qué ser constante, produciendo retrasos que los agentes deberán tener en cuenta.

El diseño permite que se implementen centros de producción tan complejos como se deseen siempre que sigan el protocolo de funcionamiento descrito más arriba. En nuestra cadena de mercados hemos implementado un tipo de centro de producción que fabrica cada uno de los productos mencionados en el apartado productos, todos ellos con un tiempo de producción igual.

2.3.8 *Agentes.*

Los agentes son los individuos que llevan a cabo las transacciones de productos en los mercados. Son el principal objeto de la simulación puesto que es su comportamiento el que queremos optimizar, si bien en ocasiones será

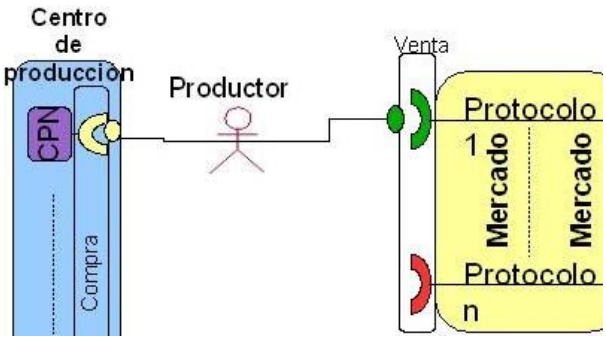
conveniente tener algunos agentes con comportamientos estándar y predecibles en alguna etapa de la cadena para poder obtener resultados concluyentes sobre los resultados conseguidos por los agentes cuyo comportamiento modelamos. En nuestra simulación los agentes cuyos resultados se muestran son los vendedores de MercaMadrid.

Los vendedores y compradores reales, como es obvio, no están especificados en ningún lugar por lo que su diseño ha sido basado en lo descrito en los distintos trabajos que hemos encontrado en la fase de investigación y en nuestra propia experiencia e intuición.

Cualquier agente podrá comprar o vender en cualquier mercado siempre y cuando sepa utilizar los protocolos que ofrece dicho mercado.

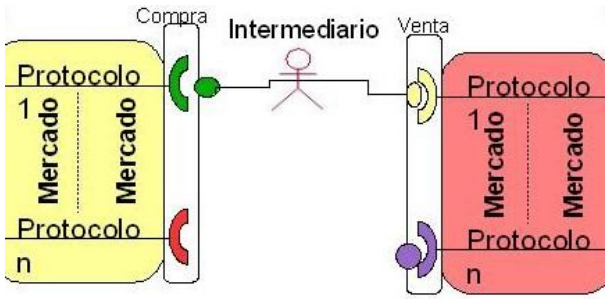
Pueden ser de tres tipos según su función:

- Productores.



  - Son los agentes que compran productos directamente a un centro de producción para luego venderlos en un mercado. Se diferencian de los demás en que su faceta de comprador no la hacen en un mercado sino en el centro de producción.

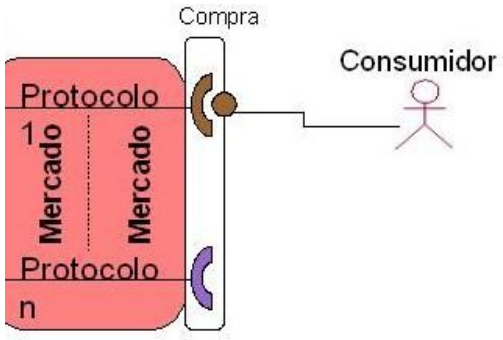
- Intermediarios.



  - Los intermediarios son agentes que compran en uno o varios mercados y venden en otro/s. Deben conocer al menos un protocolo de compra y otro de venta.

  - Los vendedores de Mercamadrid en nuestra simulación (y la mayoría en la realidad) son de este tipo, por tanto son sus resultados los que nos interesan y éstos dependerán tanto de su habilidad comprando como vendiendo.

- Consumidores.



  - Los consumidores son el final de la cadena de mercados. Son agentes que compran productos a los vendedores de MercaMadrid. Deberán conocer el protocolo de compra que se use en dicho mercado.

  - Son los que depositan su confianza en un vendedor u otro en función de varias variables, por lo tanto su comportamiento y, más concretamente, el método de elección de un vendedor u otro será de vital importancia para la simulación.

- En cierto sentido tienen una función opuesta a la de los centros de producción dado que los consumidores retiran productos de la cadena e introducen dinero.
- El comportamiento de los compradores implementados se puede ver detalladamente en el diagrama de estados de las páginas 83 y 84.

2.3.9 Visión de la confianza y la reputación en el sistema.

A la hora de diseñar la aplicación hemos tenido que decidir entre las diferentes interpretaciones que se pueden hacer de la confianza y la reputación.

En el campo de la visibilidad, descrito en [4], hemos elegido usar la aproximación consistente en tomar la confianza como privada (un agente tiene una confianza en cada uno de los otros que solamente él conoce y para que otro la conozca éste debe decírsela o vendérsela) y la reputación como privada (la reputación de un agente es general, cada agente sólo tiene una reputación y todo el mundo puede conocerla).

En lo referente a la dimensión ontológica de la confianza, mencionada también por Jordi Sabater en su tesis[4], la confianza en un vendedor por parte de un comprador se compone a partes iguales de la confianza que tiene en que los productos que entrega son de calidad y la confianza que tiene en que entrega los productos en el tiempo indicado.

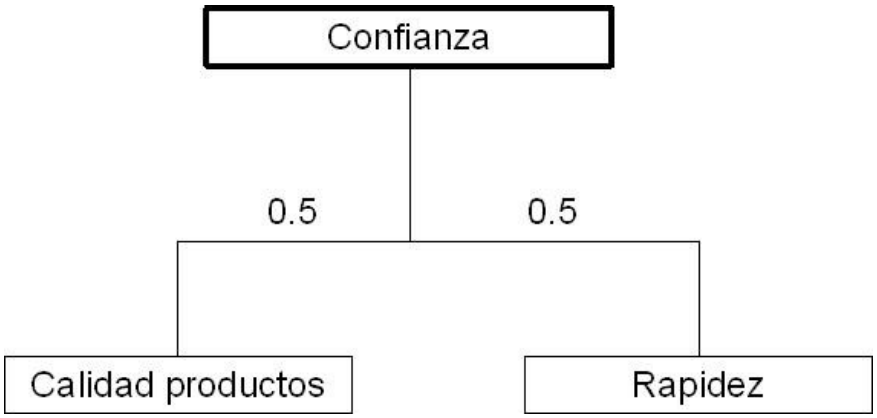


Figura 2.3.9.1 – Dimensión ontológica de la confianza.

A la hora de depositar la confianza en un vendedor u otro también influye el precio de la oferta, pero este factor no se ve reflejado en el cómputo de la confianza que hace el comprador.

Para computar la reputación de un agente hemos implementado varios métodos descritos en [18] y que aparecen explicados con todo detalle en el apartado de experimentación de esta memoria.

2.3.10 Herramientas: GUI, Gráficas y JavaDoc.

En este apartado explicamos que herramientas provee nuestro proyecto y como utilizarlas.

- *JavaDoc*

  Se ha generado documentación en formato JavaDoc del código del programa, que describe detalladamente los métodos y clases que hemos considerado más importantes para facilitar la comprensión para una posible ampliación posterior del código por parte de otros desarrolladores. Se puede encontrar en el CD del proyecto.

- *Gráficas*

  Ayudado por la herramienta RePast el programa genera gráficas que describen el comportamiento de la simulación. Se puede configurar el momento de aparición de las gráficas así como el intervalo de actualización en el formulario de configuración.

  Las gráficas de Dinero/Tiempo y Ventas/Tiempo nos muestran información sobre la evolución de los agentes y los mercados durante la simulación. Podemos apreciar el grado de éxito o fracaso que ha obtenido cada agente así como los movimientos del mercado.

  En cuanto a las gráficas de Reputación/Tiempo y Confianza/Tiempo nos muestran el desarrollo del sistema bajo

el punto de vista de la confianza y la reputación de los agentes. Podemos apreciar que agentes han sido excluidos y cuales han sido los más fiables.

- *GUI*

  La interfaz gráfica es sencilla de utilizar. Consta únicamente de dos formularios: el principal y el de configuración.

  La interfaz gráfica es específica de la cadena simulada, puesto que muestra las características propias de los agentes y los mercados. Si estas características cambiasen se debería modificar la interfaz en consecuencia.

  En el formulario principal podemos iniciar, parar o reanudar la simulación y utilizar los botones para mostrar la información de la cadena de mercados y de los agentes en cada momento.

  Es recomendable pausar la simulación cuando queramos visualizar la información de un agente en concreto. Para ello, debemos pulsar sobre el agente en la parte inferior izquierda y la información, se mostrará en un formato de tablas en la parte inferior derecha del formulario.

Figura 2.3.10.1 – Formulario principal.

En el formulario principal podemos configurar los agentes que actuarán en nuestro mercado, así como la existencia del Juez, el tipo de cómputo para la reputación y las características de las gráficas.

Para más información, consultar el manual de usuario que se encuentra en el anexo de este documento.

3 . Detalles de implementación.

3.1 CARACTERÍSTICAS DE LA APLICACIÓN

La aplicación ha sido implementada mediante el entorno de desarrollo de Eclipse [13].

El lenguaje de programación utilizado es JAVA (JDK 1.5) [14].

Para el funcionamiento de esta, es necesaria la librería repast.jar ya que se utilizan métodos de esta librería para realizar las simulaciones.

Para poder iniciar la aplicación es necesario que el equipo donde se ejecute tenga:

- La JVM (Java Virtual Machine) con la versión 1.5 ya que se disponen de tipos que no están disponibles en versiones de la JVM inferiores.
- El programa Repast. Este programa es necesario para poder realizar las simulaciones ya que es el que realiza la planificación temporal y proporciona los ticks a la aplicación para que los distintos componentes de la misma vayan interactuando.

Se ha dedicado gran esfuerzo a asegurar la fiabilidad de la aplicación, evitando problemas derivados de una mala gestión de los agentes o de los mercados.

La aplicación es mantenible, es decir, se ha diseñado de manera modular para que cualquier cambio que se pueda producir bien en la vista o bien en el modelo (también está subdividida modularmente), no afecte al otro módulo.

La aplicación, como ya se indico anteriormente, es muy portable, ya que solo necesita la máquina virtual de Java y el Repast para poder ser ejecutada desde cualquier computador.

Antes de mostrar el diseño UML de nuestra aplicación, vamos a hacer una introducción informal a grandes rasgos de cómo se estructura la misma.

Hemos intentado hacer la aplicación extensible y modificable. Para ello hemos intentado reducir el acoplamiento entre clases mediante la introducción de numerosos interfaces que describiesen secciones de su comportamiento.

De esta manera, cambiar la construcción interna de una clase no involucrará cambios en las clases que la utilicen, sino tan sólo en ella misma.

Asimismo, hemos utilizado los protocolos de compra y venta como fachada para aislar a los agentes de las implementaciones concretas de los mercados.

El hecho de utilizar muchos interfaces aumenta mucho el número de clases abstractas y parece complicar el diseño. Esto, añadido a que hemos representar el diseño en UML con el mayor nivel de detalle posible, provoca que algunos diagramas tengan una apariencia compleja.

Para facilitar la comprensión del diseño hemos hecho este pequeño esquema general.

Visión general de la cadena de mercados.

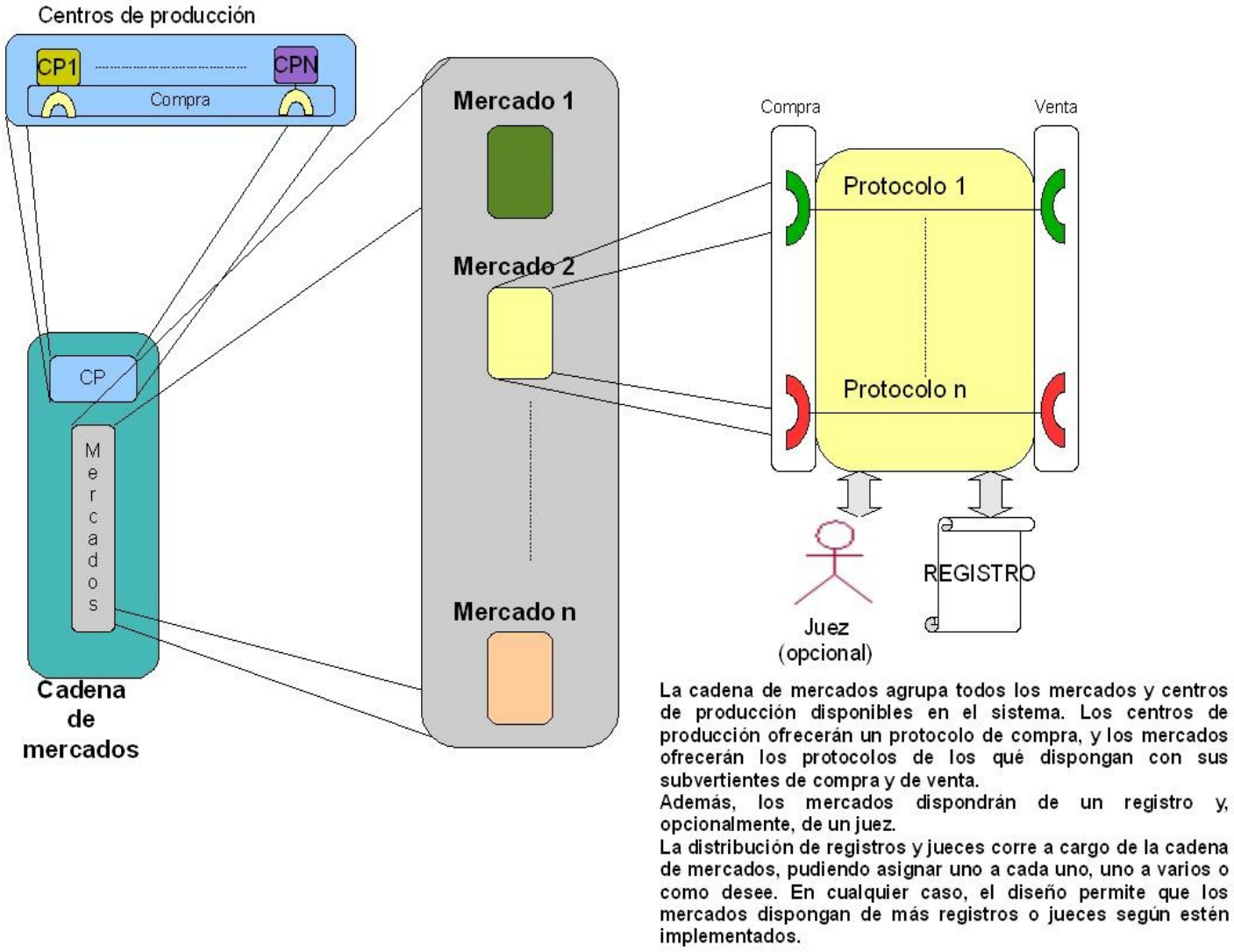


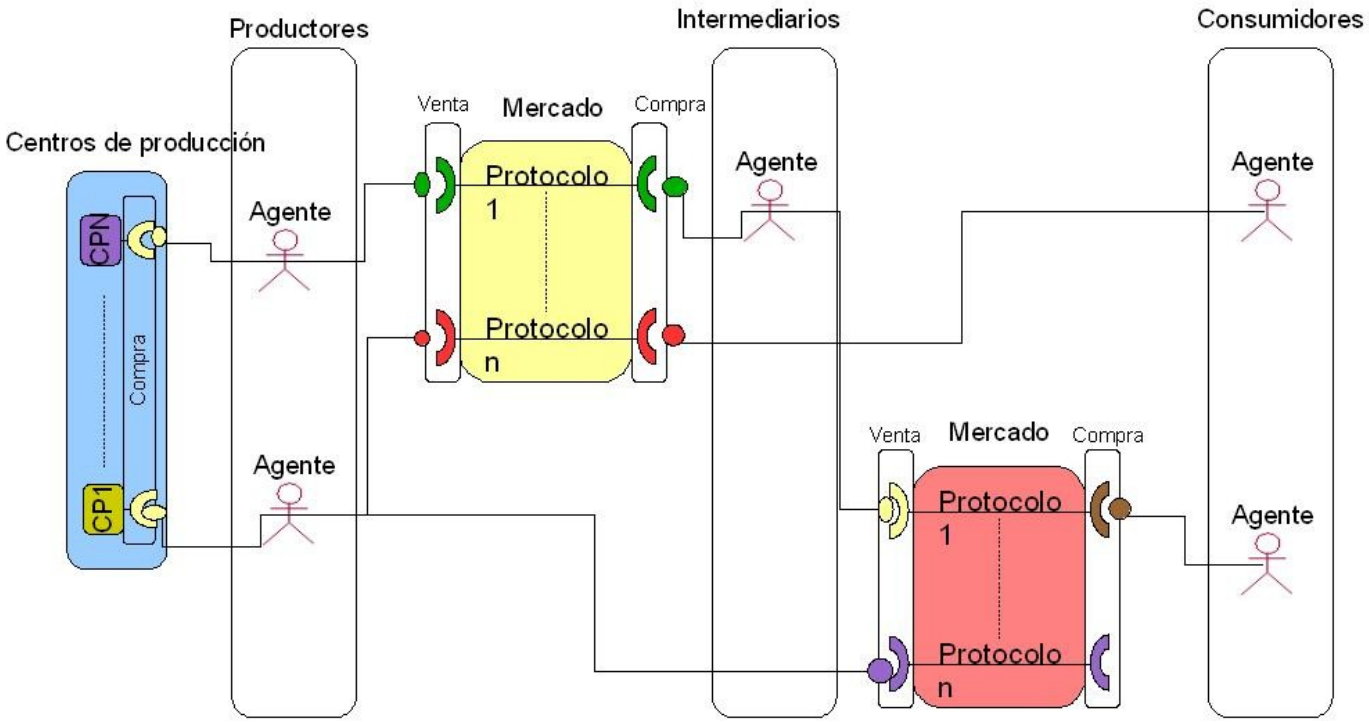
Figura 3.1.1 – Visión general de la cadena de mercados.

La clase CadenaDeMercados es el núcleo de nuestra aplicación. Ella se encarga de la configuración inicial del sistema y del funcionamiento posterior del mismo.

Hereda de la clase SimModelImpl de Repast, con lo cual representa el modelo a simular. Gracias a esto, Repast canaliza a través de esta clase los pasos

de ejecución. La clase CadenaDeMercados propaga este paso a los centros de producción y a los mercados, que a su vez los propagan a los agentes que dependan de ellos[1].

Visión de la distribución de los agentes y la topología de la cadena.



La distribución de los agentes corre a cargo de la cadena de mercados (generalmente a instancias del formulario de configuración). Ésta distribuirá los agentes asignándoles mercados en los que comprar o vender. Como se observa en el diagrama, los agentes pueden comprar en uno o varios mercados y vender también en uno o varios.
El diagrama está simplificado, la cadena de mercados puede ser arbitrariamente larga, y la complejidad de las relaciones entre éstos (dada por qué agentes actúan de puente entre los mercados) muy alta.
No existe ninguna restricción de diseño respecto a bucles (que un agente compre algo en un mercado y lo vuelva a vender en el mismo después de pasar por otros o no) dado que los propios agentes encontrarán poco competitivo vender productos que han envejecido en mercados donde hay productos nuevos.

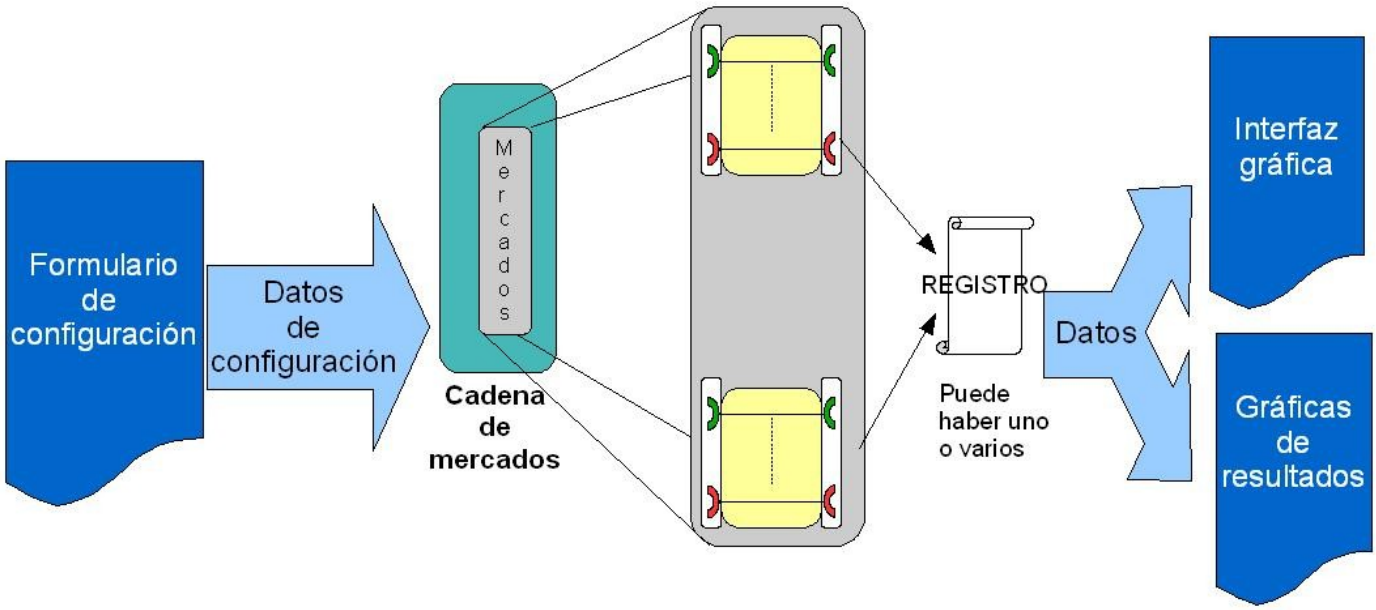
Figura 3.1.2 - Visión de la distribución de los agentes y la topología de la cadena.

Este diagrama muestra como los agentes se relacionan con los mercados a través de los protocolos que estos proporcionen. Estos protocolos serán las fachadas que hemos mencionado anteriormente. Estas fachadas, a su vez, estarán

1 Si un agente pertence a más de un mercado solo uno le propaga el paso. CadenaDeMercados se encarga de notificar a los mercados si deben notificar el paso o no a un agente al añadirlo.

enmascaradas por dos interfaces, uno de compra y uno de venta. Este enmascaramiento evita que un agente acceda a datos y/o operaciones que no debe conocer. Esto aparece explicado con más detalle en el comentario de los diagramas UML.

La interacción con el usuario.



Para configurar la cadena de mercados disponemos de un formulario de configuración que permite elegir la forma de configurarla. Los datos recogidos por este formulario indican a la cadena de mercados que tipo de agentes ha de poner en los mercados.
A la hora de mostrar los datos y resultados de la simulación, el/los registro/s de los mercados proporcionan acceso tanto a la GUI como a las gráficas a todos los mercados, agentes y transacciones realizadas en el mercado, así como a los datos de reputación que almacenan.
Para reducir el acoplamiento del diseño, hemos utilizado interfaces para aislar la configuración de las clases de implementación concretas.

Figura 3.1.3 – Interacción con el usuario.

El diagrama muestra las relaciones de la capa de vista con la de modelo. Todas estas relaciones se harán también a través de interfaces que faciliten acceso total de lectura a los datos de agentes y mercados a la GUI y a las gráficas de resultados.

Una vez vista la estructura general del programa, detallamos el diseño completo del mismo utilizando el lenguaje UML.

3.2 DISEÑO UML

El diseño del simulador del mercado se ha realizado con el programa Bouml versión 2.3.0. Este software es no comercial y se puede descargar gratuitamente de la página web del autor [15].

Para observar con mayor claridad los diagramas, se pueden ver con más detalle en el archivo del diseño.

3.2.1 Diagramas de Clases:

Las clases de la aplicación se han agrupado en doce paquetes distintos según la relación existente entre ellas.

Para el desarrollo de la aplicación se ha utilizado el patrón de arquitectura de software Modelo Delegado. Este tipo de arquitectura es una variante del patrón Modelo Vista Controlador en el que la Vista y el Controlador se fusionan en lo que es el Delegado manteniéndose separado el modelo.

- Modelo: es el encargado de gestionar los datos la aplicación.

- Delegado: es el encargado de gestionar como los datos se muestran al usuario a través de la interfaz gráfica de la aplicación, así como captar las acciones realizadas por el usuario e indicar al modelo los cambios que tiene que realizar.

El paquete GUI es el que realiza la función del Delegado, mientras que los demás paquetes forman el modelo de la aplicación.

Ahora vamos a pasar a comentar detenidamente cada uno de los paquetes:

- ***PAQUETE CADENADEMERCADOS***

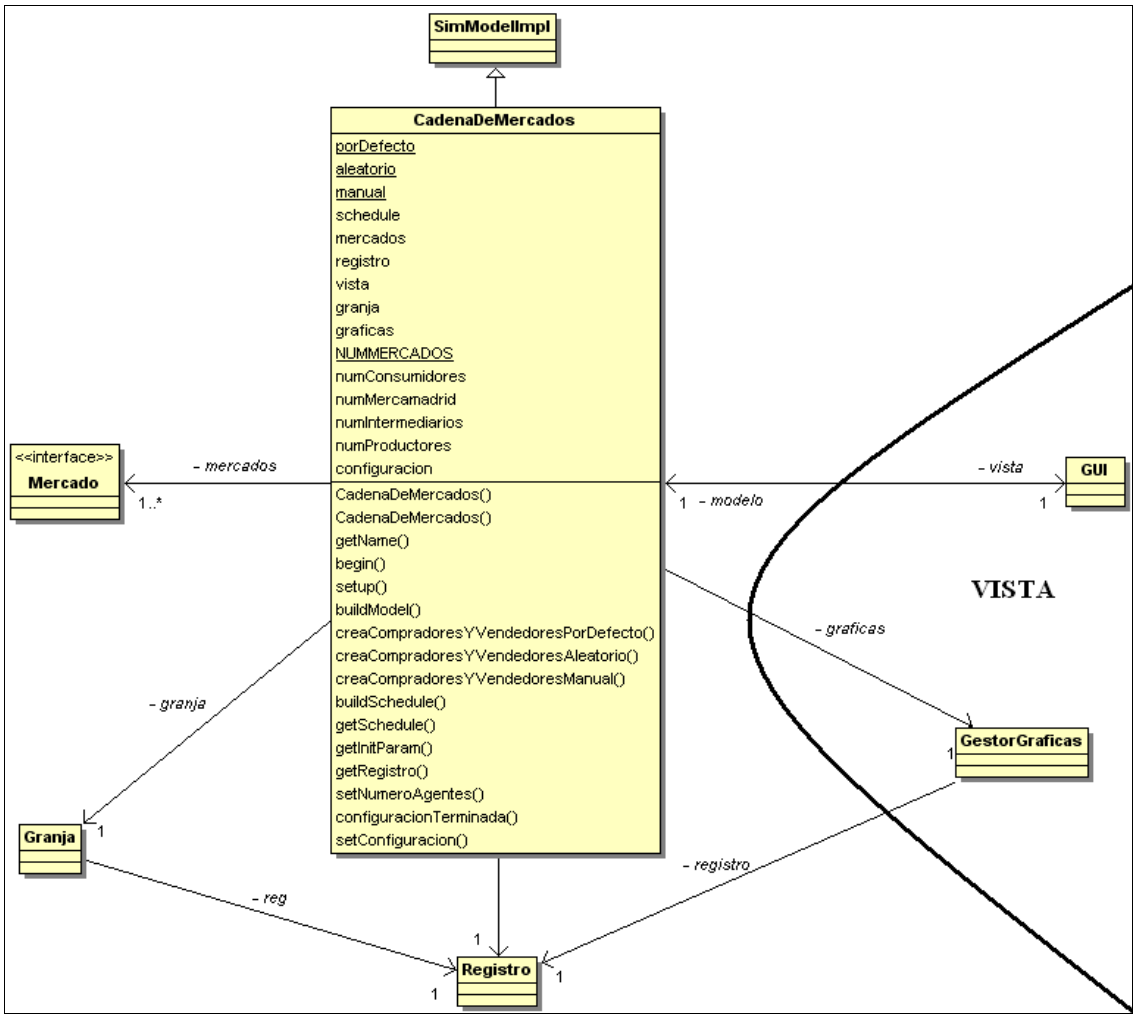


Figura 3.2.1.1: Diagrama de Clases del Paquete cadenaDeMercados.

Está formado por la clase CadenaDeMercados que hereda de la clase SimModelImpl perteneciente a la librería repast.jar.

La clase CadenaDeMercados contiene un Array de tamaño NUMMERCADOS de objetos de la clase Mercado, un objeto de la clase Registro, otro de un centro de producción (en este caso Granja), otro objeto de la clase GestorGráficas y un puntero a la GUI.

En esta clase actúa como el modelo de la aplicación. Aquí es donde se crean los mercados para la simulación, se introducen los diferentes agentes compradores y vendedores, se crea el registro (si hubiese que crear más de uno se crearían aquí tambien, aunque los mercados pueden tener sus propios registros) que llevará el

control de todo lo ocurrido; los centros de producción para la generación de los productos, y un objeto GestorGraficas para la creación de gráficas para mostrar los resultados de la simulación.

El objeto registro que se crea en esta clase, como ya se comentó anteriormente, en nuestra cadena es único para toda la aplicación. Se crea en esta clase, y es utilizado por los mercados para dejar constancia en él de lo ocurrido en la ejecución de la aplicación.

En esta clase también se implementan los métodos necesarios para pasarle los ticks (pasos de simulación generados por repast) a todos los elementos de la aplicación para que se pueda producir la simulación.

- ***PAQUETE MERCADO***

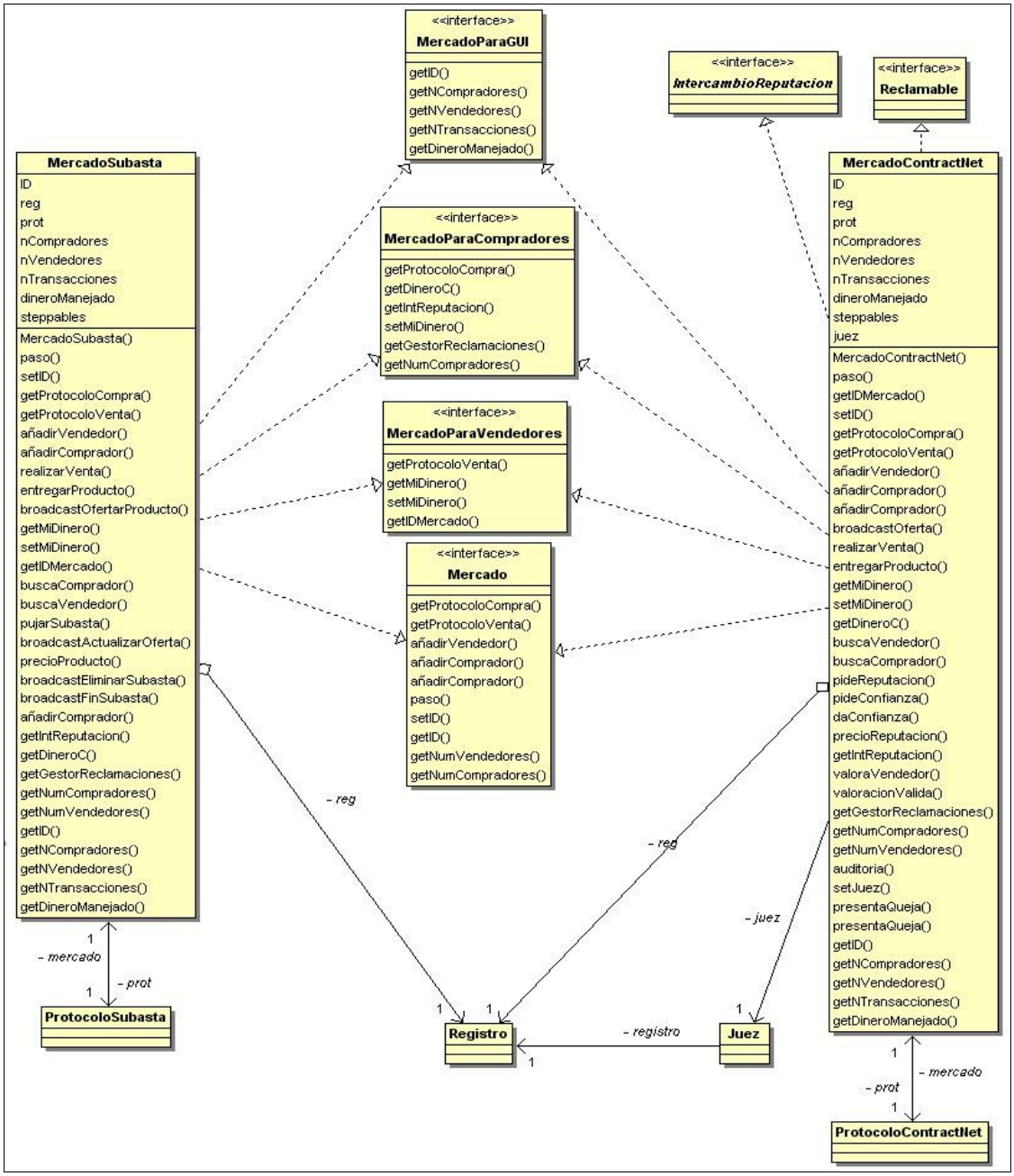


Figura 3.2.1.2: Diagrama de Clases del Paquete mercado.

Está compuesto por las clases MercadoContractNet y MercadoSubasta que implementan las interfaces Mercado, MercadoParaCompradores y

MercadoParaVendedores que encapsulan las operaciones de compra y las operaciones de venta respectivamente, para aislar a los agentes de operaciones que no deben conocer; y MercadoParaGUI que encapsula las operaciones para obtener datos para mostrarlos al usuario.

La clase MercadoContractNet implementa la interface Reclamable, que encapsula las operaciones para que los compradores de este mercado puedan presentar una queja sobre un vendedor, y la interface IntercambioReputacion que encapsula las operaciones para el intercambio de confianza y reputación.

La clase MercadoSubasta contiene un objeto de la clase ProtocoloSubasta, que será el protocolo que ofrece a los agentes, y un puntero al registro de la aplicación donde registrará sus eventos.

La clase MercadoContractNet contiene un objeto de la clase ProtocoloContractNet, un puntero al registro de la aplicación y un objeto de la clase Juez.

Tanto en la clase MercadoSubasta como en la MercadoContractNet se implementan las operaciones correspondientes a los dos tipos de protocolos implementados en la aplicación, el Contract Net y la Subasta Inglesa. Además en MercadoContractNet implementa métodos para las operaciones de intercambio de reputación y de penalización de agentes.

Para introducir nuevos mercados que funcionen con un protocolo diferente de compraventa bastaría que el nuevo mercado implementara las interfaces Mercado, MercadoParaCompradores, MercadoParaVendedores y MercadoParaGUI e incluir los nuevos métodos necesarios para que las transacciones se realizaran en relación al nuevo protocolo.

- ***PAQUETE PROTOCOLOS***

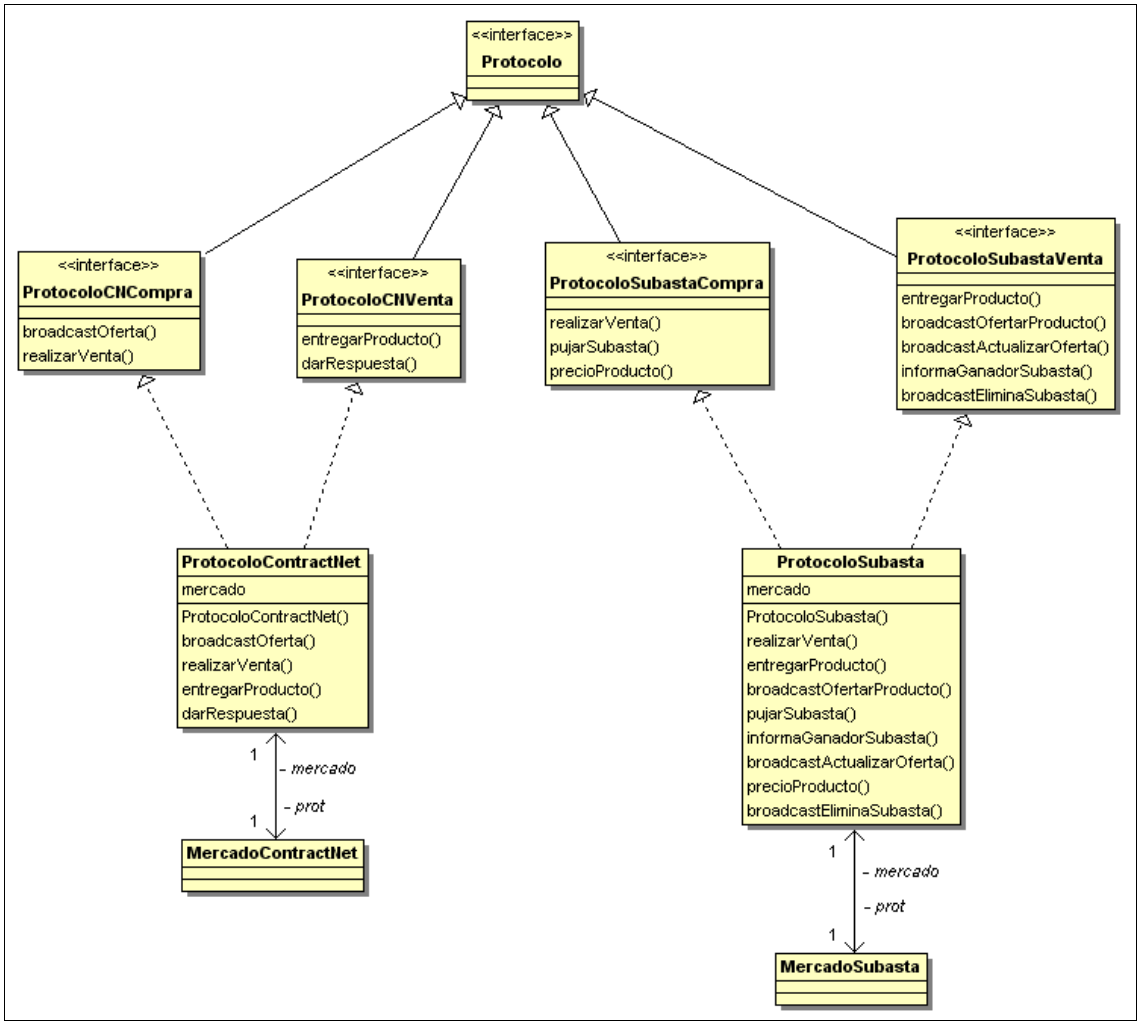


Figura 3.2.1.3: Diagrama de Clases del Paquete protocolos.

Está compuesto por las clases:

- La clase ProtocoloContractNet que implementa las interfaces ProtocoloCNCompra y ProtocoloCNVenta que encapsulan las operaciones de compra y venta correspondientes al protocolo de compraventa Contract Net.

- La clase ProtocoloSubasta que implementa las interfaces ProtocoloSubastaCompra y ProtocoloSubastaVenta que encapsulan las operaciones de compra y venta correspondientes al protocolo de compraventa Subasta Inglesa.

- Las interfaces ProtocoloCNCompra, ProtocoloCNVenta, ProtocoloSubastaCompra y ProtocoloSubastaVenta heredan de la interface Protocolo indicando que son versiones de protocolos.

- La clase ProtocoloContractNet contiene un objeto de la clase MercadoContractNet y la clase ProtocoloSubasta contiene un objeto de la clase MercadoSubasta.

- La clase ProtocoloContractNet contiene las operaciones de copra y de venta con las que se realiza el protocolo Contract Net. Cada una de estas operaciones de compraventa del protocolo se pasa a su objeto MercadoContractNet donde se realizan dichas operaciones.

- La clase ProtocoloSubasta tiene las operaciones correspondientes de compra y venta para la implementación de una Subasta Inglesa. Cada una de estas operaciones de compraventa del protocolo se pasa a su objeto MercadoSubasta donde se realizan dichas operaciones.

- La interface IntercambioReputacion contiene las operaciones para solicitar y recibir confianza y reputación de otros agentes, así como operaciones para que un comprador valore a un vendedor modificando su reputación. Para ello el comprador podrá realizar una valoración positiva (+1), negativa (-1), o neutra (0) del vendedor. Estas valoraciones se podrán extender si se implementan nuevas formas de valorar a los vendedores.

- La interface Reclamable contiene las operaciones para que un comprador pueda realizar una queja sobre un vendedor por una transacción realizada con él.

Tanto la clase ProtocoloContractNet como la clase ProtocoloSubasta siguen el patrón de diseño fachada. Estas dos clases separan la interacción entre los distintos tipos de agentes y los mercados en los que actúan.

Así todas las transacciones que los agentes realizan con otros agentes del mismo mercado por medio de un protocolo de compraventa-determinado, pasan por la clase Protocolo correspondiente en la que se invoca al método del mercado que corresponde a la transacción.

Se pueden crear nuevos tipos de protocolos compraventa siguiendo la estructura de los ya existentes, es decir, creando una interface para las operaciones de venta, otra para las de compra y una nueva clase para el nuevo protocolo que implemente las dos interfaces anteriormente mencionadas y

que utilice un mercado que utilice ese protocolo de compraventa para realizar sus transacciones.

- ***PAQUETE REGISTRO***

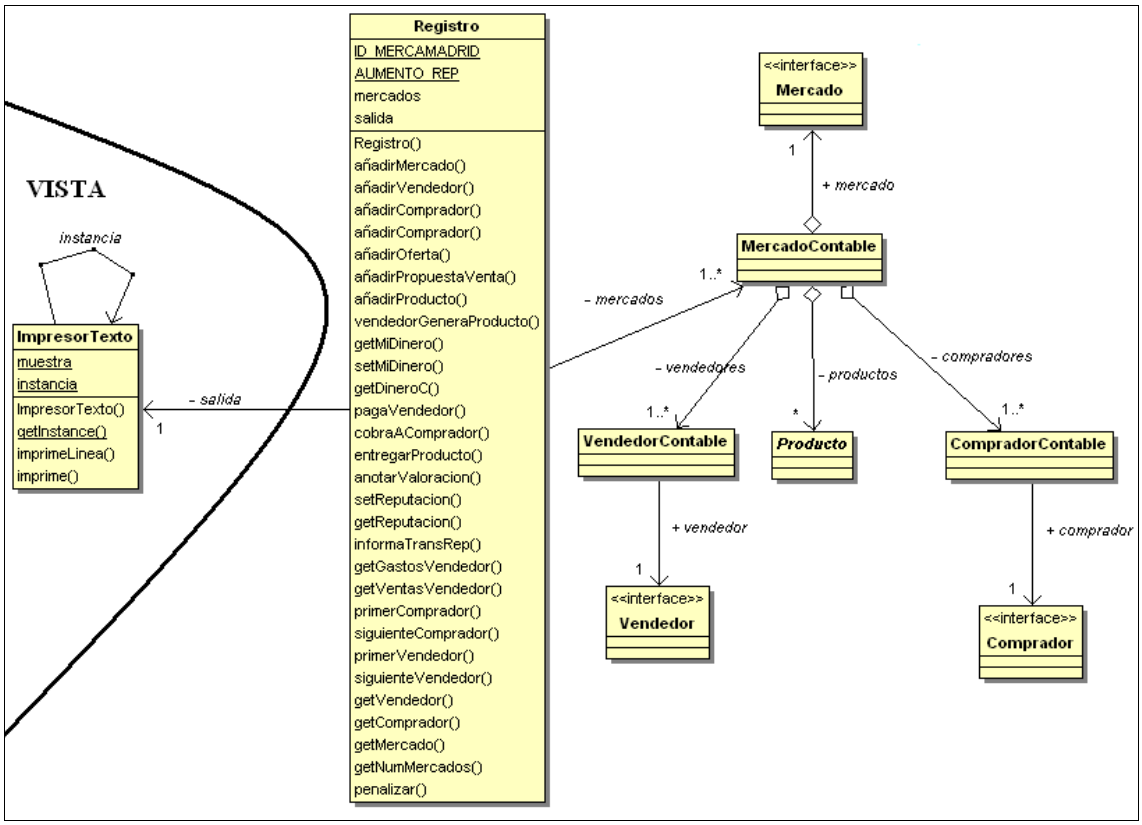


Figura 3.2.1.4: Diagrama de Clases del Paquete registro.

Está formado por la clase Registro que tiene tres clases internas (MercadoContable, CompradorContable, VendedorContable) para gestionar la contabilidad de los diferentes mercados.

- CompradorContable contiene un objeto de la clase Comprador, y el dinero que tiene.
- VendedorContable contiene un objeto de la clase Vendedor, el dinero que tiene, los gastos que tiene, las ventas realizadas, las existencias de productos y su reputación.

- MercadoContable contiene un objeto de la clase Mercado, un ArrayList de objetos VendedorContable, otro de CompradorContable y otro de Productos.

Con estas tres clases, se simula la contabilidad de los compradores y de los vendedores, es decir, el dinero que manejan entre todos los componentes de la aplicación y la reputación de los mismos. Toda esta información queda reflejada en el registro.

La clase Registro contiene un ArrayList de MercadosContables y un puntero a la clase ImpresorTexto, que pertenece a la parte de la Vista de la aplicación, para mostrar información por consola de las transacciones que quedan reflejadas en el registro. Como cada uno de estos mercados está compuesto por todos los compradores, vendedores, y todos los productos que hay en dicho mercado, el registro lleva cuenta de todos los agentes de la aplicación y de todos los productos que se intercambian entre ellos.

En la clase registro queda almacenada cualquier acción que se produce en cualquier mercado de los existentes como la creación de nuevos compradores y vendedores, la creación de nuevos productos, de ofertas de compra, de subastas, de contabilidad con respecto a las transacciones realizadas, etc.

La clase Registro utiliza el patrón de comportamiento Iterador. Se usa debido a que el registro contiene los compradores y vendedores de cada mercado y necesita recorrerlos y acceder a ellos para realizar distintas operaciones.

(Ver los diagramas de secuencia para observar el funcionamiento del registro).

- ***PAQUETE JUEZ***

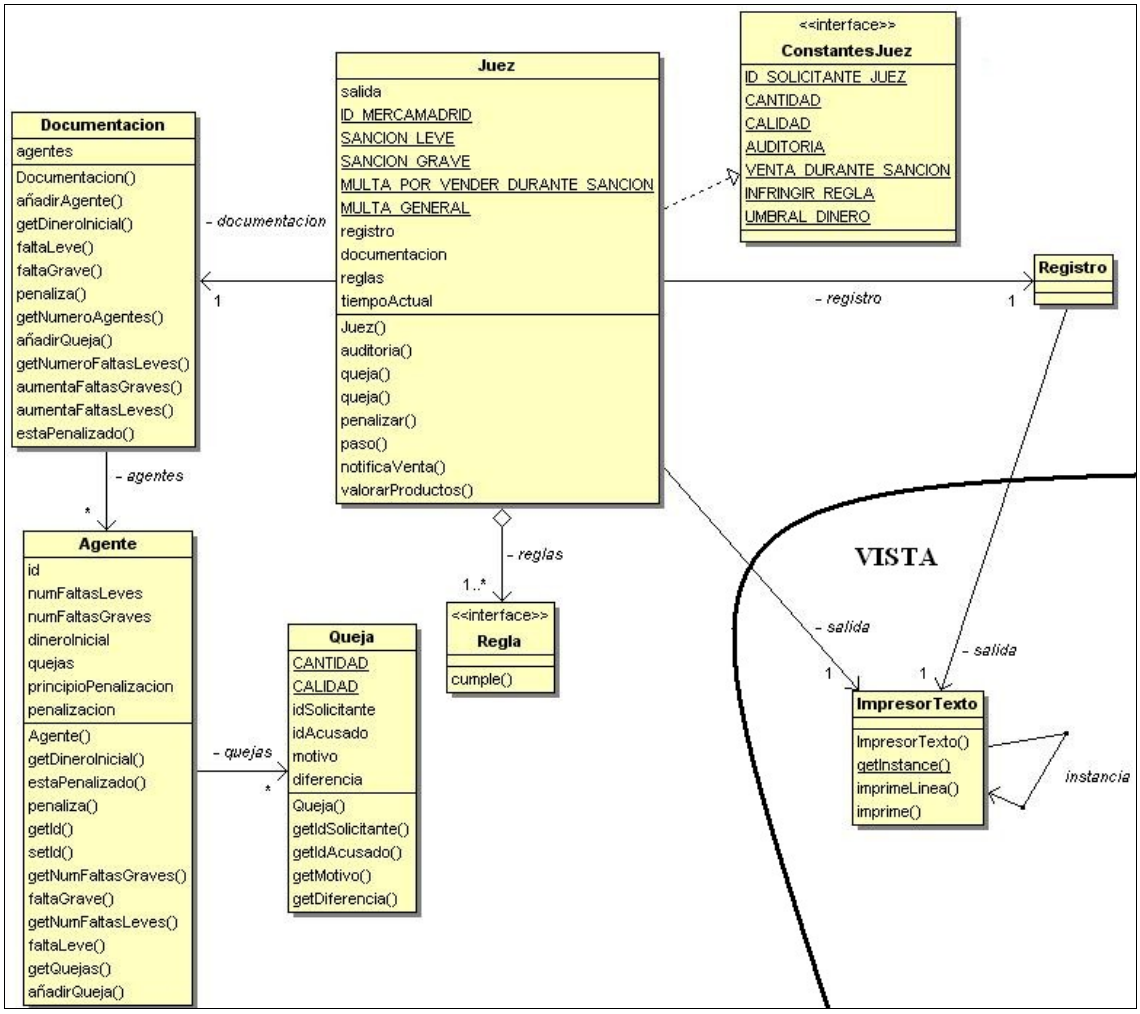


Figura 3.2.1.5: Diagrama de Clases del Paquete juez.

Está formado por las clases Juez, Documentación, Queja y Regla.

La clase Juez implementa la interface ConstantesJuez que contiene una serie de constantes para indicar el motivo de la penalización, la dureza de las sanciones a aplicar, y los umbrales de penalización.

Esta clase Juez contiene un puntero al registro de la aplicación (lo utilizará para acceder a los datos de transacciones anteriores y agentes), un objeto de la clase Documentación, un ArrayList de objetos de la clase Regla un puntero al registro de la aplicación y otro puntero a la clase ImpresorTexto para la impresión por consola.

La clase Documentación tiene un ArrayList de objetos Agente que es una clase interna a Documentación. A su vez cada objeto de la clase Agente tiene un ArrayList de objetos de la clase Queja. Esta clase interna agente guarda las penalizaciones de los vendedores, el número de faltas que tienen, las quejas que los compradores les han hecho etc. Así, en Documentación se le pasan a los objetos de la clase Agente las faltas que van cometiendo, las quejas recibidas, y sus correspondientes penalizaciones.

La clase Juez es la encargada de realizar auditorias de lo que un vendedor de Mercamadrid ha declarado que ha vendido, para comprobar si se comete alguna infracción. Así, cada cierto tiempo, el juez elige un vendedor del mercado al azar y comprueba sus cuentas para ver si todo está correcto.También lleva la cuenta de las quejas (a través de la clase Queja) que los compradores realizan al mercado por alguna transacción realizada con un vendedor que no le ha sido satisfactoria, almacenándolas por medio de la clase Documentación. El juez estudiará la queja y decidirá si es necesario penalizar o no al vendedor acusado.

(Para más claridad ver los diagramas de secuencia y actividades Realizar_Auditoría, Realizar_Queja y Penalizar).

- ***PAQUETE COMPRADOR***

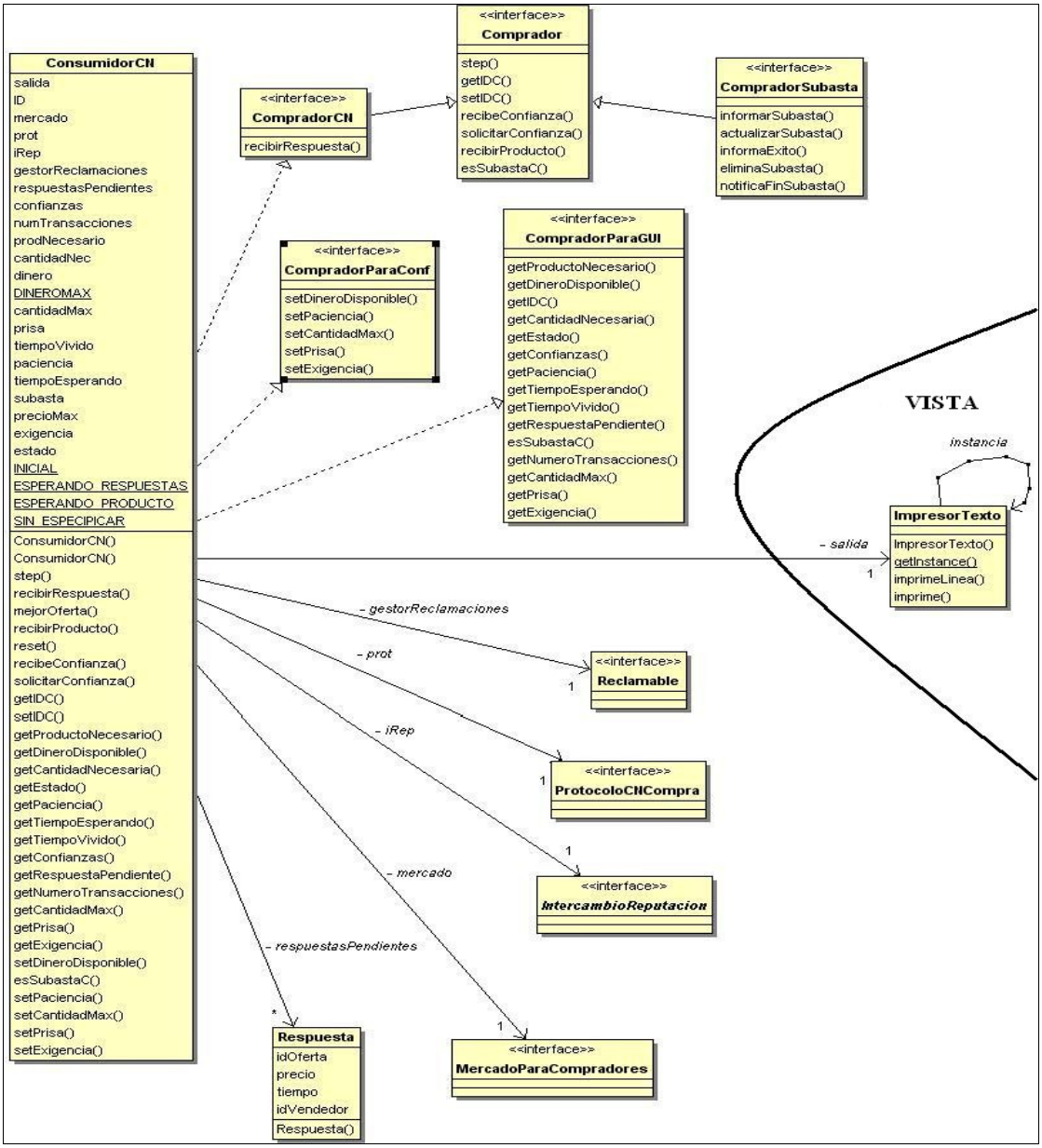


Figura 3.2.1.6: Diagrama de Clases del Paquete comprador.

Está formado por la clase ConsumidorCN que implementa la interface CompradorCN que encapsula las operaciones del protocolo Contract Net que realiza el comprador,la interface CompradorParaGUI que tiene las operaciones para obtener los datos de un Comprador y mostrarlos al usuario y la interface

CompradorParaConf que contiene las operaciones para colocar los atributos introducidos por el usuario al comprador.

La interface CompradorCN y la interface CompradorSubasta heredan de la interface Comprador que encapsula las operaciones que puede realizar cualquier comprador de los dos protocolos existentes.

Cada objeto de la clase ConsumidorCN contiene un objeto ProtocoloCNCompra, un objeto IntercambioReputacion para las transacciones de confianza y reputación, un puntero a la clase MercadoParaCompradores que es el mercado al que pertenece, un ArrayList de objetos de la clase Respuesta que es una clase interna a ConsumidorCN, un objeto Reclamable para realizar las quejas sobre algún Vendedor y un puntero a la clase ImpresorTexto para la impresión por consola.

Cada objeto de la clase Respuesta que contiene el comprador tiene almacenadas las ofertas de compra que recibe el comprador cuando solicita un producto. Así en estos objetos se almacena el vendedor que hizo la oferta, el precio de la transacción y el tiempo transcurrido.

La clase ConsumidorCN implementa todas las acciones que puede realizar un agente del tipo Consumidor (sólo realiza operaciones de compra por medio del protocolo Contract Net, las operaciones referentes al intercambio de confianza y reputación y las relacionadas con la reclamación sobre un vendedor).

(Ver los diagramas de secuencia y actividades para observar como se realizan las transacciones por parte de los compradores).

- ***PAQUETE VENDEDOR***

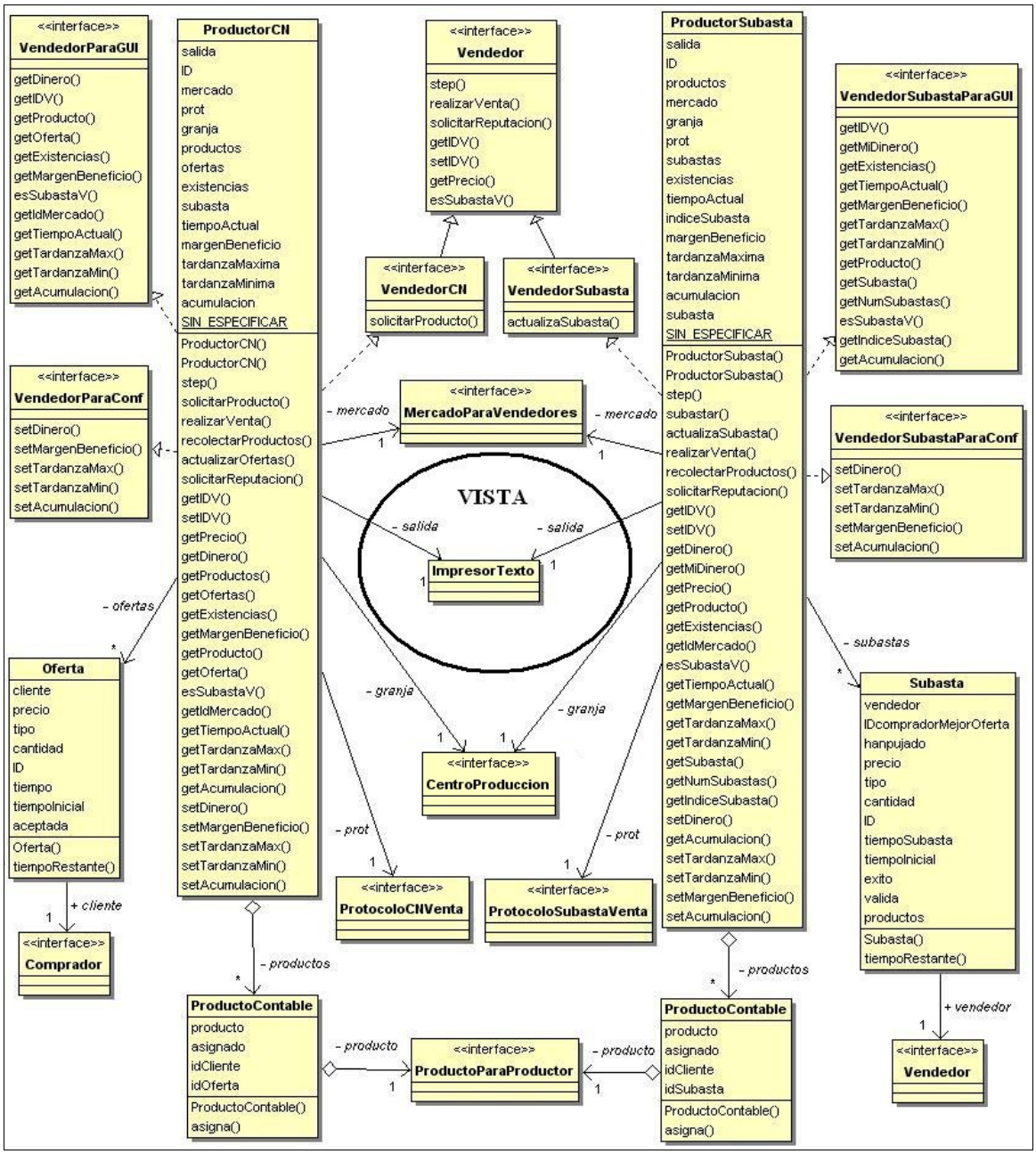


Figura 3.2.1.7: Diagrama de Clases del Paquete vendedor.

Está compuesto principalmente por las clases ProductorCN y ProductorSubasta.

La clase ProductorCN implementa la interface VendedorCN que encapsula las operaciones de venta del protocolo Contract Net, la interface VendedorParaGUI

que tiene las operaciones que devuelven los datos de un Vendedor del tipo Contract Net y poder mostrarlos al usuario y la interface VendedorParaConf que contiene las operaciones para colocar los atributos introducidos por el usuario al vendedor.

La clase ProductorSubasta implementa la interface VendedorSubasta que encapsula las operaciones de venta del protocolo Subasta, la interface VendedorSubastaParaGUI que encapsula las operaciones que devuelven los datos de un Vendedor del tipo Subasta y poder mostrarlos al usuario y la interface VendedorSubastaParaConf que contiene las operaciones para colocar los atributos introducidos por el usuario al vendedor.

A su vez las interfaces VendedorCN y VendedorSubasta heredan de Vendedor que lleva las operaciones comunes a todos los vendedores independientemente del protocolo que implementen.

La clase ProductorCN contiene un puntero a un objeto MercadoParaVendedores que es el mercado donde actúa, un puntero a un objeto CentroProduccion que es el centro de generación de productos existente en la aplicación, un objeto de ProtocoloCNVenta que es el protocolo que utiliza para vender y un puntero a la clase ImpresorTexto para mostrar información por consola.

Además tiene dos clases internas para modelar las transacciones que va a realizar que son Ofertas que contiene un objeto Comprador y ProductoContable que contiene un objeto ProductoParaProductor.

La clase ProductorCN tiene un ArrayList de objetos de la clase Oferta donde va almacenando las ofertas de venta que va realizando a los compradores. Estas ofertas contienen el Comprador al que se le hizo, el precio, el producto, la cantidad de productos y cuanto tiempo lleva vigente la oferta. También tiene otro ArrayList de objetos de la clase ProductoContable. Estos objetos son para asignar un producto del vendedor a una oferta realizada y al comprador para que una vez que un comprador acepte la oferta por esos productos el vendedor no los ofrezca a nadie más antes de entregárselos al comprador.

La clase ProductorSubasta contiene un puntero a un objeto MercadoParaVendedores que es el mercado donde actúa, un puntero a un objeto CentroProduccion que es el centro de generación de productos existente en la aplicación, un objeto de ProtocoloSubastaVenta que es el protocolo que utiliza para vender y un puntero a la clase ImpresorTexto para mostrar información por consola.

Al igual que ProductorCN, contiene dos clases internas para las transacciones que el VendedorSubasta va a realizar. Estas son Subasta que contiene un objeto Vendedor, y ProductoContable que contiene un objeto ProductoParaProductor.

La clase ProductorSubasta contiene un ArrayList de objetos de la clase Subasta donde se almacenan las subastas que el vendedor va creando. Estas subastas llevan almacenadas el comprador que hizo la última puja, el dinero de la puja, el tiempo que lleva la subasta etc. Los objetos de la clase ProductoContable son para asignar un producto del vendedor a una subasta realizada y al comprador que hizo la mejor puja para que una vez que un se termine la subasta, esos productos el vendedor no los ofrezca a nadie más antes de entregárselos al comprador.

De este modo, tanto la clase ProductorSubasta como la clase ProductorCN implementan las operaciones correspondientes a la venta utilizando sus respectivos protocolos.

(Ver los diagramas de secuencia y actividades para observar como se realizan las transacciones por parte de los vendedores).

- ***PAQUETE VENDEDORCOMPRADOR***

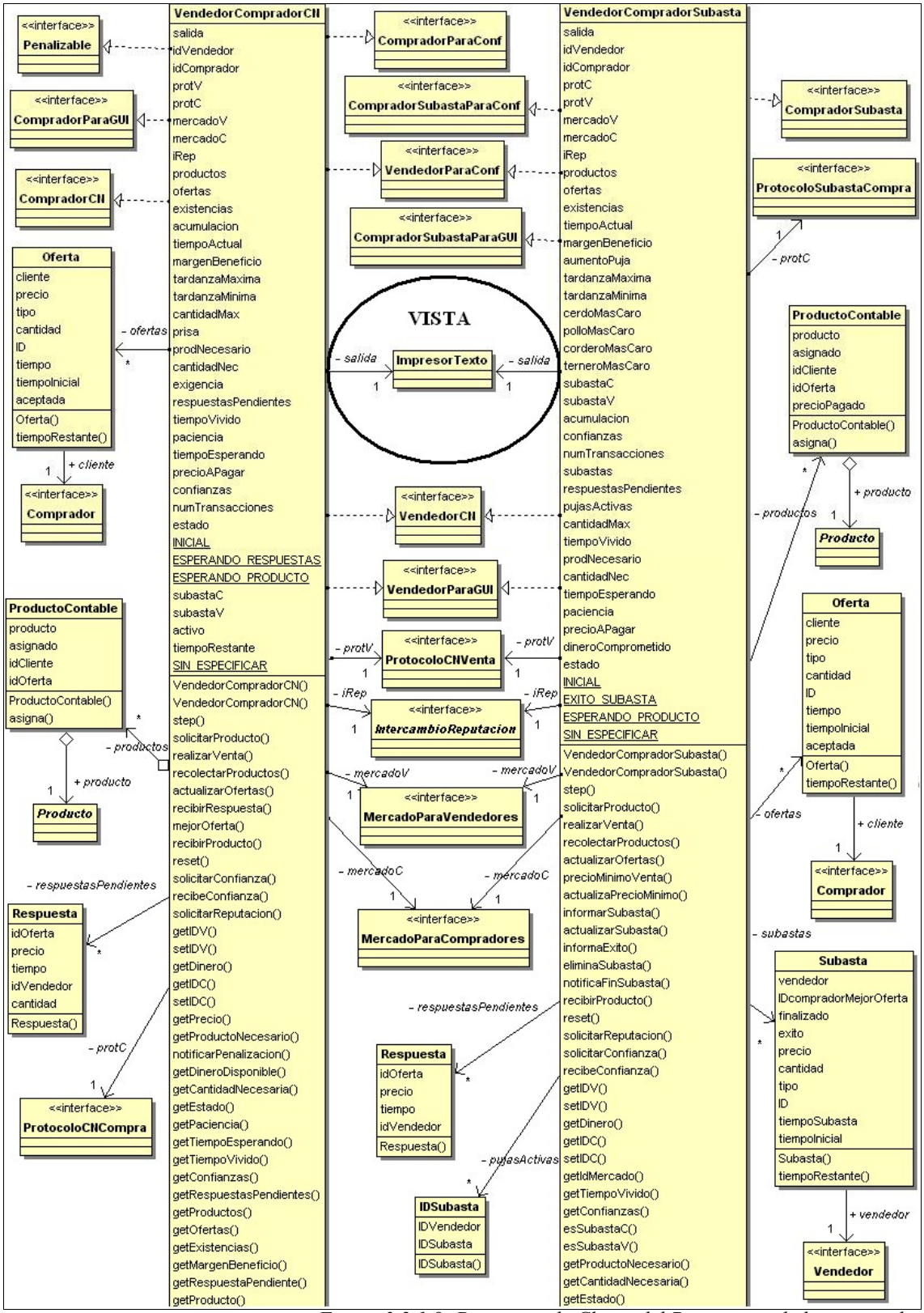


Figura 3.2.1.8: Diagrama de Clases del Paquete vendedorcomprador.

Está compuesto por las clases VendedorCompradorCN y la clase VendedorCompradorSubasta.

La clase VendedorCompradorCN implementa las interfaces CompradorCN y VendedorCN que encapsulan las operaciones de compra y venta respectivamente del protocolo Contract Net; las interfaces CompradorParaGUI y VendedorParaGUI que encapsulan las operaciones que devuelven la información del agente referente a sus ventas y a sus compras; las interfaces CompradorParaConf y VendedorParaConf que contienen las operaciones para colocar los atributos introducidos por el usuario al vendedor-comprador, y la interface Penalizable que contiene las operaciones para que el agente pueda ser penalizado por haber cometido alguna infracción.

La clase VendedorCompradorSubasta implementa las interfaces CompradorSubasta y VendedorCN que encapsulan las operaciones de compra referentes al protocolo Subasta y las operaciones de venta referentes al protocolo Contract Net respectivamente; las interfaces CompradorSubastaParaGUI y VendedorParaGUI que encapsulan las operaciones que devuelven la información del agente referente a sus ventas y a sus compras, y las interfaces CompradorSubastaParaConf y VendedorParaConf que contienen las operaciones para colocar los atributos introducidos por el usuario al vendedor-comprador, y la interface Penalizable que contiene las operaciones para que el agente pueda ser penalizado por haber cometido alguna infracción.

La clase VendedorCompradorCN contiene un objeto ProtocoloCNVenta y un objeto ProtocoloCNCompra que son los protocolos que utiliza el agente para vender y para comprar respectivamente; un puntero a un objeto MercadoParaCompradores y otro a un objeto MercadoParaVendedores que son los mercados donde actúa el agente; un objeto IntercambioReputacion para las transacciones de confianza y reputación y un puntero a la clase ImpresorTexto para mostrar información por consola.

Esta clase también tiene tres clases internas para modelizar las transacciones. Estas son Oferta que contiene un objeto Comprador, Respuestas y ProductoContable que contiene un objeto ProductoParaProductor.

Así, VendedorCompradorCN contiene un ArrayList de objetos de la clase Oferta donde va almacenando las ofertas de venta que va realizando a los compradores. Estas ofertas contienen el Comprador al que se le hizo, el precio, el producto, la cantidad de productos y cuanto tiempo lleva vigente la oferta. Tiene otro ArrayList de objetos de la clase ProductoContable. Estos objetos son para asignar un producto del vendedor a una oferta realizada y al comprador para que una vez

que un comprador acepte la oferta por esos productos el vendedor no los ofrezca a nadie más antes de entregárselos al comprador. También tiene un ArrayList de objetos de la clase Respuesta donde guarda las respuestas que obtiene de los vendedores cuando solicita un producto. En estas respuestas se almacena el vendedor que le hizo la propuesta de venta, el precio de la misma y el tiempo transcurrido.

La clase VendedorCompradorSubasta contiene un objeto ProtocoloCNVenta y un objeto ProtocoloSubastaCompra que son los protocolos que utiliza el agente para vender y para comprar respectivamente; un puntero a un objeto MercadoParaCompradores y otro a un objeto MercadoParaVendedores que son los mercados donde actúa el agente; un objeto IntercambioReputacion para las transacciones de confianza y reputación y un puntero a la clase ImpresorTexto para mostrar información por consola.

La clase contiene también cinco clases internas para modelizar las transacciones. Estas son Oferta que contiene un objeto Comprador, Respuestas, Subasta que contiene un objeto Vendedor, IDSubasta y ProductoContable que contiene un objeto ProductoParaProductor.

Así, VendedorCompradorCN contiene un ArrayList de objetos de la clase Oferta donde va almacenando las ofertas de venta que va realizando a los compradores. Estas ofertas contienen el Comprador al que se le hizo, el precio, el producto, la cantidad de productos y cuanto tiempo lleva vigente la oferta. Tiene otro ArrayList de objetos de la clase ProductoContable. Estos objetos son para asignar un producto del vendedor a una oferta realizada y al comprador para que una vez que un comprador acepte la oferta por esos productos el vendedor no los ofrezca a nadie más antes de entregárselos al comprador. Otro ArrayList de objetos de la clase Respuesta donde guarda las respuestas que obtiene de los vendedores cuando solicita un producto. En estas respuestas se almacena el vendedor que le hizo la propuesta de venta, el precio de la misma y el tiempo transcurrido. Otro ArrayList de objetos de la clase Subasta donde se almacenan las subastas existentes en la aplicación. Estas subastas llevan almacenadas el comprador que hizo la última puja, el dinero de la puja, el tiempo que lleva la subasta etc. También tiene un ArrayList de objetos de la clase IDSubasta que almacena las subastas que aun están activas y por tanto puede pujar en ellas.

Con todo esto, la clase VendedorCompradorCN va a implementar todas las operaciones tanto de compra como de venta pertenecientes al protocolo Contract Net, mientras que la clase VendedorCompradorSubasta implementa las operaciones correspondientes a la compra del protocolo Subasta y a las operaciones de venta correspondientes al protocolo Contract Net.

Los agentes VendedorCompradorSubasta compran y venden de diferente manera ya que van a realizar operaciones de compra a los productores que tiene como forma de venta la subasta, y van a vender los productos obtenidos por el protocolo Contract Net porque sus compradores van a ser los vendedores del mercado mayorista que van a comprar en Contract Net.

- ***PAQUETE CENTROS_PRODUCCION***

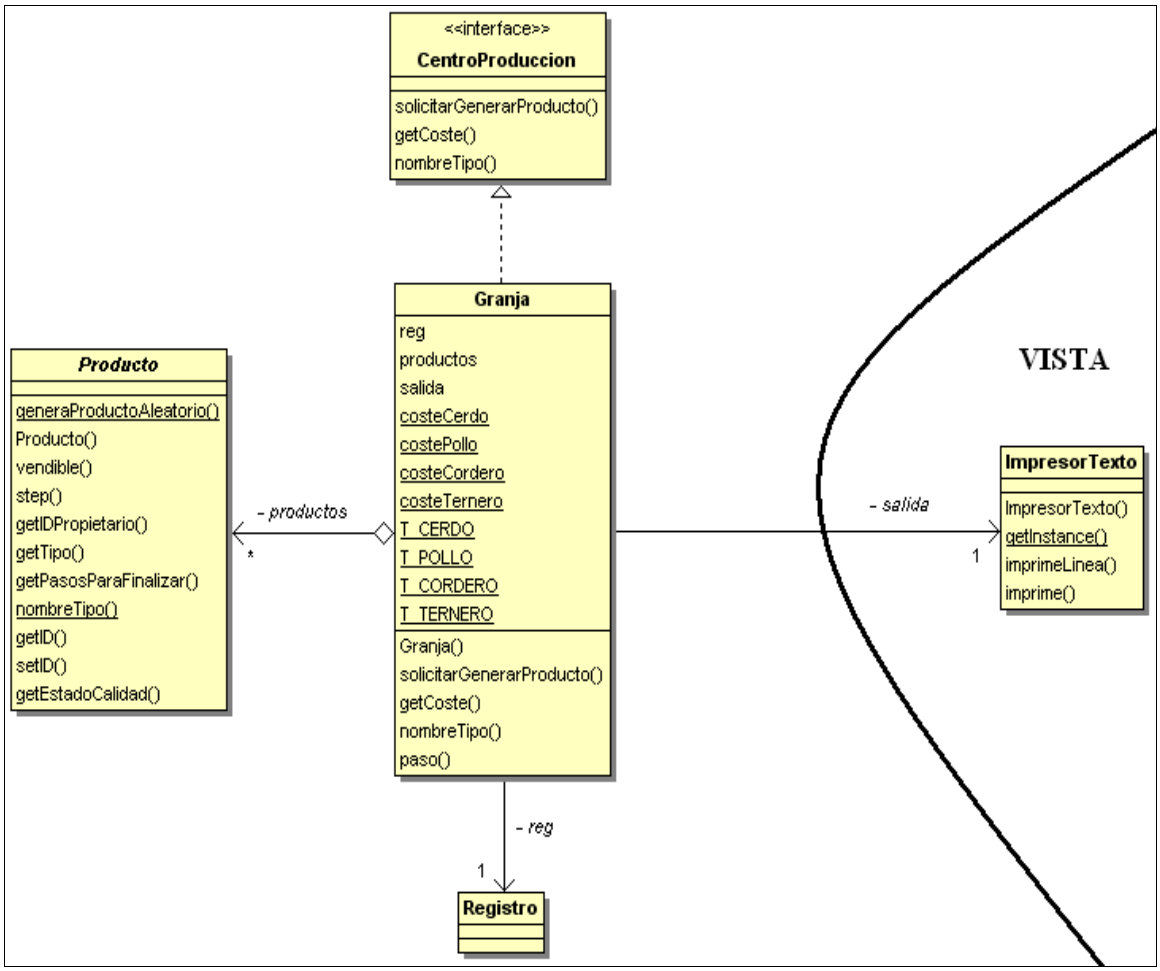


Figura 3.2.1.9: Diagrama de Clases del Paquete centros_produccion.

Está compuesto por la clase Granja que implementa la interface CentroProduccion.

La clase Granja contiene un ArrayList de productos, un puntero al registro de la aplicación (objeto de la clase Registro) y otro a la clase ImpresorTexto, que pertenece a la parte de la Vista de la aplicación, para mostrar información por consola de la creación de los nuevos productos.

En la clase Granja cuando un productor solicita la generación de un determinado producto, este se empieza a crear, y se añade al registro general de la aplicación que lleva el control de todo lo que ocurre en los mercados.

La granja sigue el patrón factoría ya que en la clase Granja es donde se crean las instancias a los diferentes tipos de producto que se crean en la aplicación.

(Ver Diagramas de Secuencia y Actividades Generar_Producto).

- ***PAQUETE PRODUCTO***

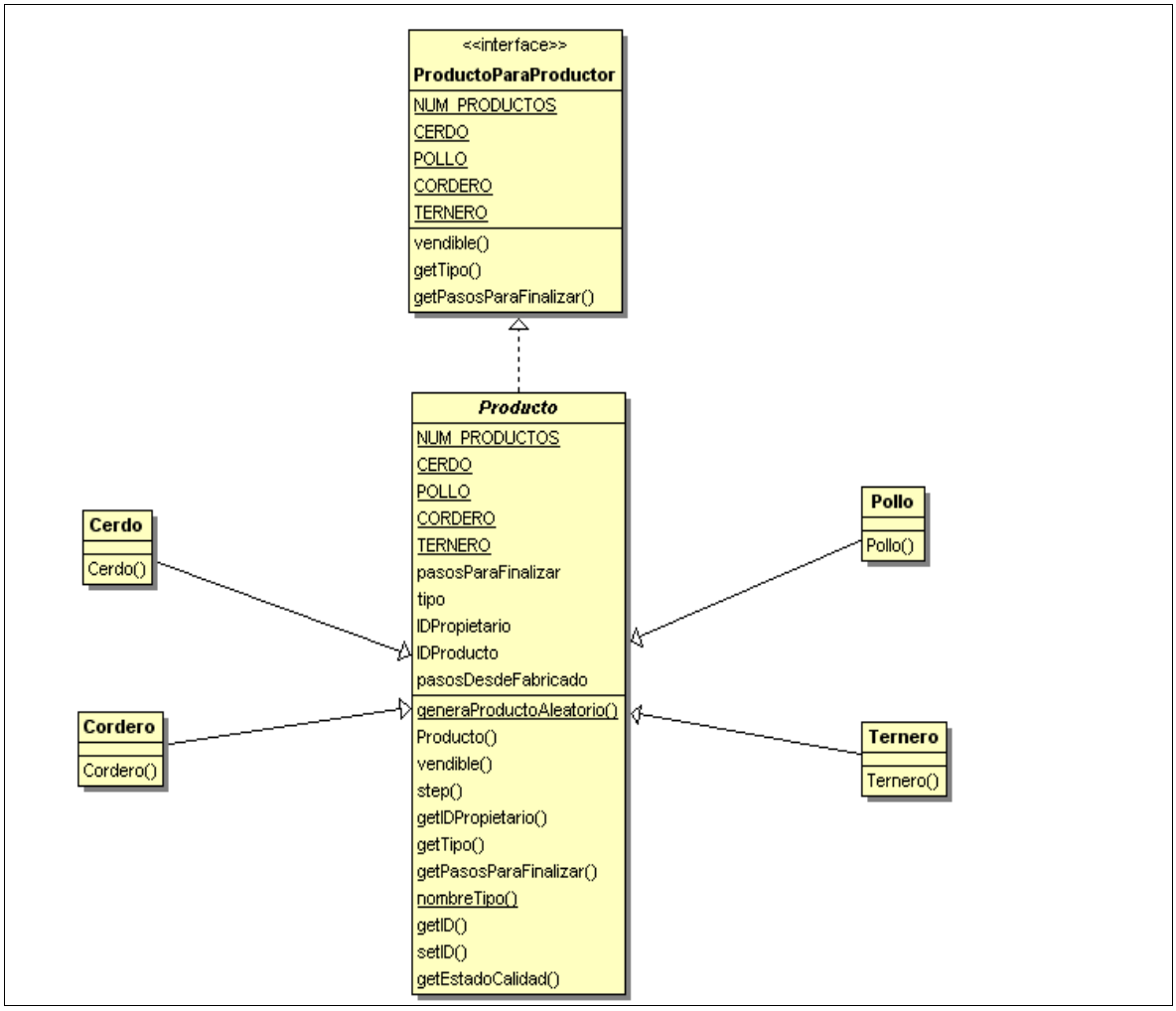


Figura 3.2.1.10: Diagrama de Clases del Paquete producto.

Está compuesto por la clase Producto que implementa la interface ProductoParaProductor.

La clase Producto corresponde a los productos que se generan en el centro de producción a petición de un productor. La clase dispone de unas constantes para darle un identificador a cada uno de los diferentes tipos de producto. Cada producto lleva asociado el tiempo que falta para que se complete su producción (pasosParaFinalizar), el vendedor al que pertenece y el tiempo que tiene el producto desde que se fabricó (pasosDesdeFabricacion).

Las clases Cordero, Ternero, Pollo y Cerdo corresponden a cada uno de los diferentes productos que se crean en la aplicación. Estas clases heredan de Producto para poder especificar características propias de cada uno de los tipos de producto.

La estructura de este paquete está hecha de esta manera para que a la hora de añadir cualquier producto nuevo que posea unas características específicas, baste con crear una nueva clase con el nuevo producto con sus características que herede de producto y añadir un nuevo identificador para ese producto.

- ***PAQUETE GUI***

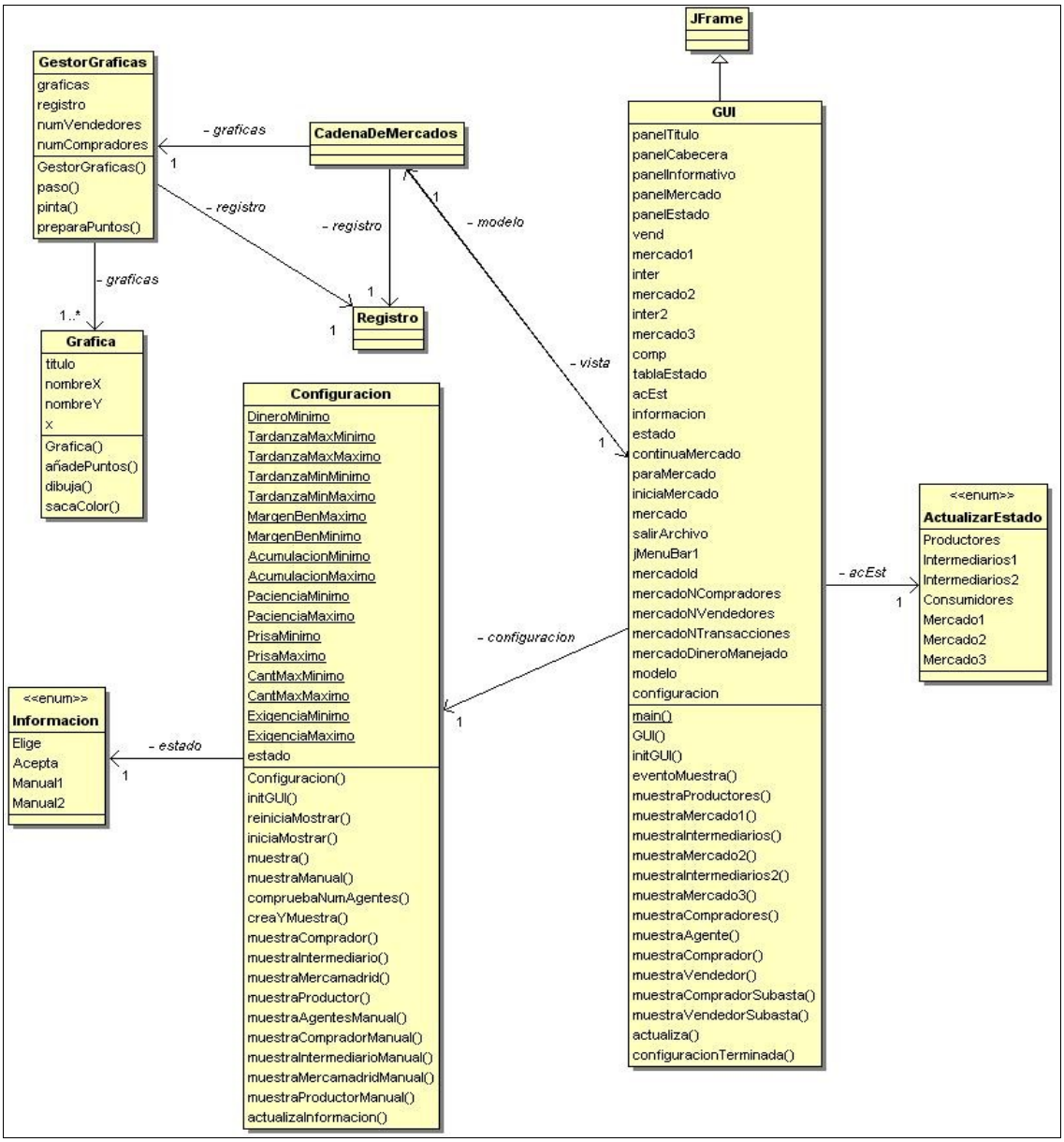


Figura 3.2.1.11: Diagrama de Clases del Paquete GUI.

Está compuesto por:

- La clase GUI que hereda de JFrame y que representa la interfaz gráfica de la aplicación.

- La clase Configuración que representa la interfaz gráfica para que el usuario pueda configurar la aplicación.

- La clase GestorGraficas que contiene un puntero al registro de la aplicación y un Array de objetos de la clase Grafica que son las gráficas que se generan por la simulación de los mercados.

- La clase ImpresorTexto para mostrar información por consola de las transacciones que se realizan en la aplicación.

La clase GUI contiene:

- El objeto de la clase CadenaDeMercados (clase principal donde se crea el registro de la aplicación, los mercados, los centros de producción, etc.) que actúa como modelo de la aplicación. En este objeto CadenaDeMercados se crea un objeto de la clase GestorGraficas donde se crean las gráficas que se muestran al usuario para observar los datos de la simulación. Las distintas gráficas generadas son: Dinero del Vendedor/Tiempo, Reputación del Vendedor/Tiempo, Confianza del Vendedor/Tiempo y Ventas del Vendedor/Tiempo.

- El objeto de la clase Configuracion que es la vista del panel de configuración del usuario para que el usuario seleccione el número de agentes por cada mercado y las características de estos. Además utiliza un tipo enumerado llamado Informacion para llevar en cuenta en cada momento, la información de configuración que tiene que mostrar al usuario.

- Utiliza un tipo enumerado llamado ActualizarEstado para llevar la información del tipo de agente que el usuario ha solicitado conocer.

Cuando se inicia la aplicación y el usuario elige la opción de iniciar, la GUI llama a la vista de configuración para que elija el tipo de configuración de los diferentes mercados. Una vez que queda configurado se inicia la simulación. Durante ésta, el usuario puede seleccionar diferentes opciones en la interfaz gráfica de la aplicación, para que la GUI le muestre la información correspondiente a cualquiera de los mercados y de los agentes que intervienen en cada uno de estos. Así se mostrará la información correspondiente a los mercados, es decir, el número de agentes del mercado, el dinero manejado y el número de transacciones realizadas. O bien, la información correspondiente a los agentes de los mercados, es decir, los productos que tienen, su dinero, las ofertas o subastas que puedan tener, etc.

Para facilitar el intercambio de información de la vista con los agentes, se poseen clases Transfer. La vista solicita a un determinado agente la información de alguna de sus transacciones, entonces éste rellena el objeto Transfer y se lo entrega a la vista para que pueda recoger los datos y mostrarlos al usuario.

La clase ImpresorTexto sigue el patrón Singleton, es decir, se crea una única instancia de esta clase en toda la aplicación. Cuando en algún momento se necesita el objeto ImpresorTexto, se comprueba que no haya ninguna instancia del mismo. Si no la hay la crea, mientras que si ya existe le pasa la instancia existente.

- ***PAQUETE TRANSFER***

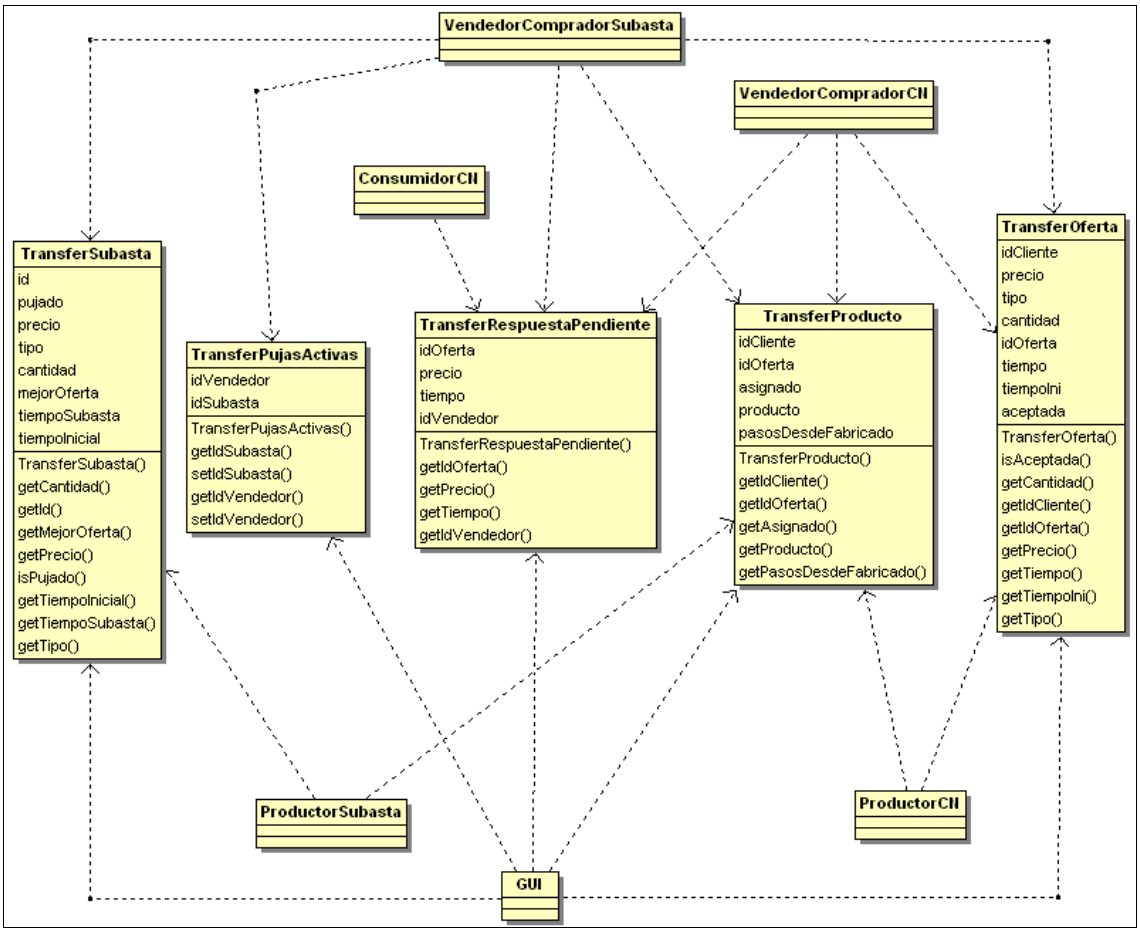


Figura 3.2.1.12: Diagrama de Clases del Paquete transfer.

Contiene cinco clases Transfer para el manejo de la información de los agentes y de sus respectivas transacciones con la GUI de la aplicación. Las clases Transfer existentes son las siguientes: TransferOferta, TransferSubasta, TransferProducto, TransferRespuestaPendiente y TransferPujasActivas.

La clase GUI utiliza las cinco clases transfer ya que muestra la información de todos los tipos de agentes.

Cada tipo de agente utiliza las clases transfer que corresponden a las transacciones que realizan.

Este tipo de objetos son utilizados por la vista para facilitar el paso de la información de los agentes a esta. Cuando la vista necesita mostrar información de un determinado tipo de agente, la vista invoca al método especifico del agente para obtener el objeto transfer de dicha información y con este objeto obtener los datos necesarios y mostrarlos al usuario a través de la interfaz gráfica.

La relación entre los distintos paquetes se puede ver por medio de los siguientes diagramas:

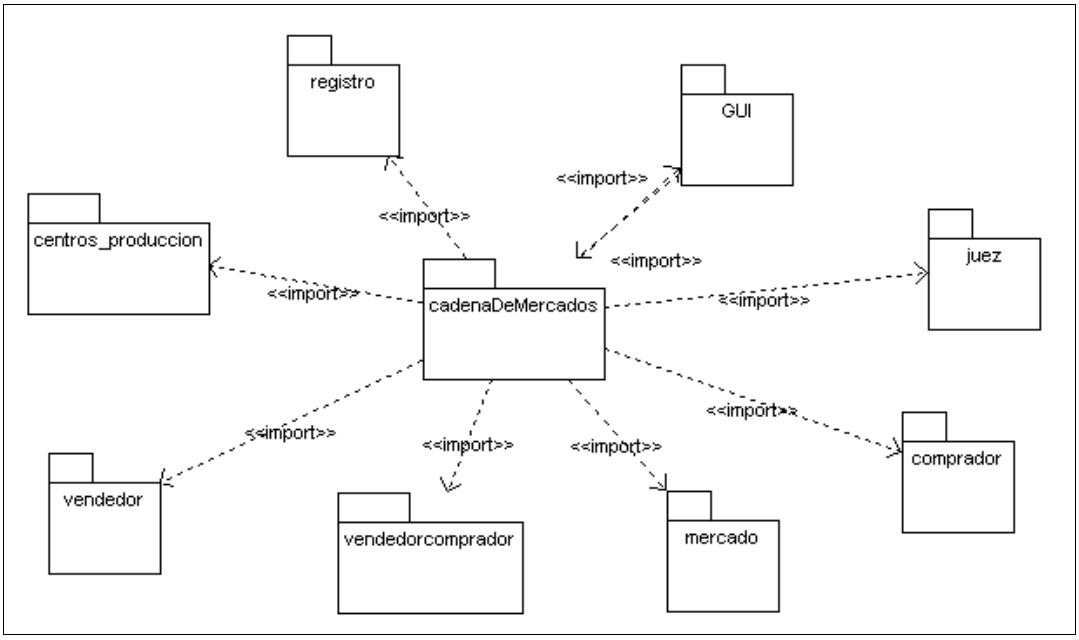


Figura 3.2.1.13: Diagrama de Paquetes.

El paquete cadenaDeMercados importa a los siguientes paquetes:

- GUI ya que el modelo contiene la vista principal de la aplicación y crea el gestor de gráficas para la generación de las diferentes gráficas con los resultados.
- registro ya que en el modelo es donde se crea el registro único de la aplicación.
- centros_produccion ya que en el modelo se crean los centros de producción de la aplicación, en este caso, una granja.
- mercado, comprador, vendedor y vendedorcomprador ya que en el modelo es donde se crean los tres mercados de la aplicación, así como los compradores, vendedores y compradores-vendedores de estos tres mercados.
- juez ya que en el modelo se crea el juez que va a regir el mercado de MercaMadrid.

El paquete cadenaDeMercados es importado por los siguientes paquetes:

GUI ya que aquí se crea el modelo de la aplicación (patrón Modelo-Delegado).

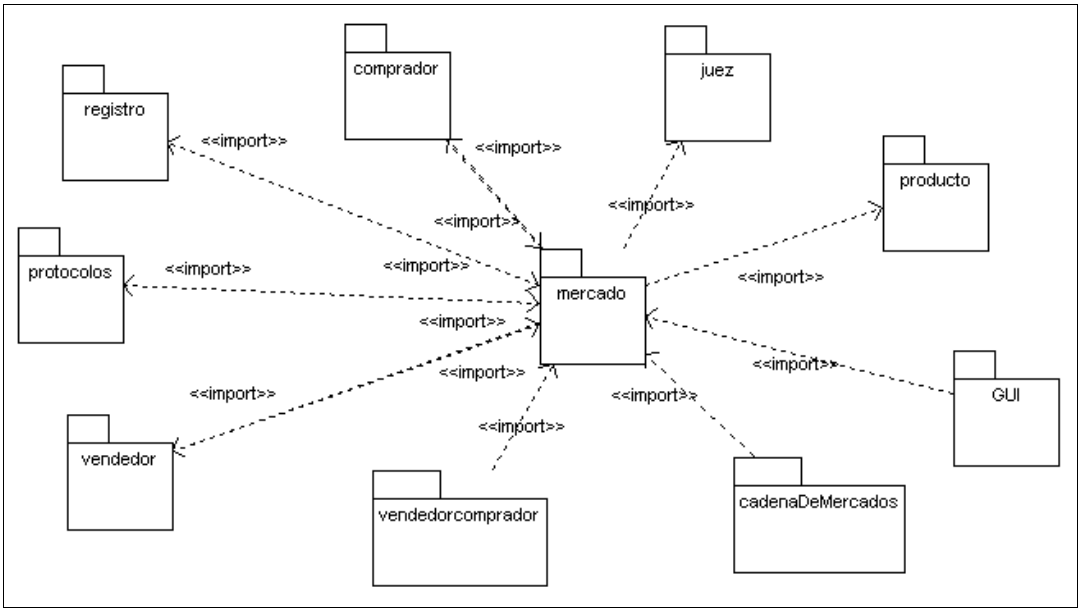


Figura 3.2.1.14: Diagrama de Paquetes.

El paquete mercado importa a los siguientes paquetes:

- comprador y vendedor ya que en los mercados actúan los compradores y los vendedores realizando operaciones de compraventa entre ellos.

- protocolos ya que cada mercado tiene su protocolo dependiendo de cómo se realizan las operaciones en dicho mercado.
- juez ya que uno de los tres mercados contiene un juez para que actúe en caso de que algún vendedor cometa una infracción.
- registro para ir añadiendo en el registro las transacciones que se producen en los mercados.
- producto ya que los mercados implementan métodos que los utilizan y aunque se podrían haber utilizado objetos del tipo Object (estos métodos solo los usan para pasarlos como parámetro de otro método de otro paquete), el ponerlos hace que el código sea más claro.

El paquete mercado es importado por los siguientes paquetes:

- cadenaDeMercados ya que en éste es donde se crean los tres mercados de la aplicación.
- comprador, vendedor y vendedorcomprador ya que estos contienen el mercado en el que actúan.
- protocolos ya que cada uno de éstos contiene un mercado que utiliza ese protocolo para realizar las operaciones de compraventa.
- registro ya que utiliza los mercados para llevar cuenta de éstos y de las transacciones que se realizan en ellos.
- GUI ya que utiliza los mercados para mostrar al usuario por medio de la vista de la aplicación, la información de cada uno de ellos.

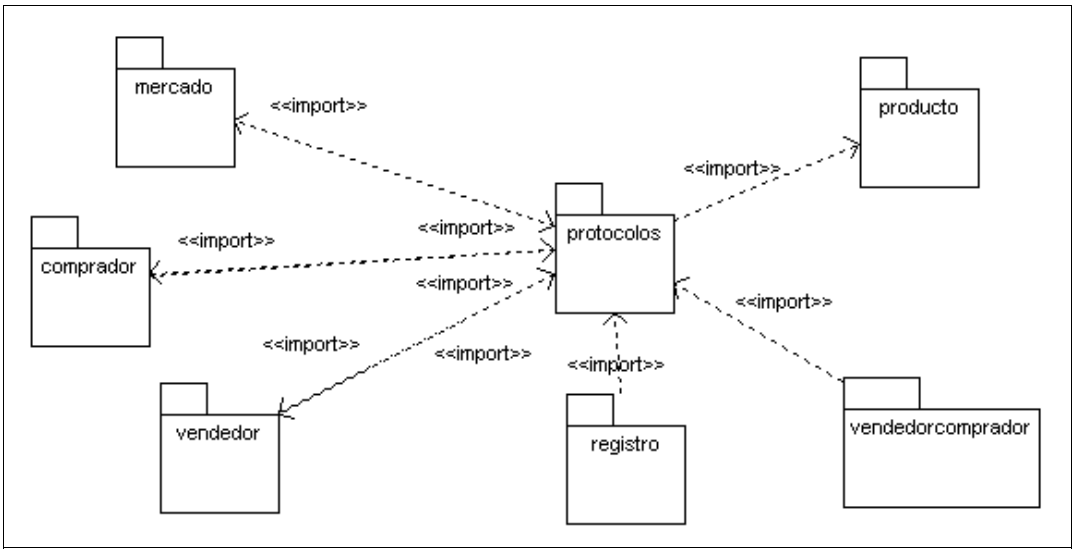


Figura 3.2.1.15: Diagrama de Paquetes.

El paquete protocolos importa a los siguientes paquetes:

- mercado ya que cada uno de los dos protocolos contiene un mercado que utiliza ese protocolo para realizar las operaciones de compraventa.
- comprador, vendedor y producto debido a que los protocolos implementan métodos que los utilizan y aunque se podrían haber utilizado objetos del tipo Object (estos métodos solo los usan para pasarlos como parámetro de otro método de otro paquete), el ponerlos hace que el código sea más claro.

El paquete protocolos es importado por los siguientes paquetes:

- mercado ya que cada uno de estos crea un protocolo dependiendo de cómo se realizan las operaciones en dicho mercado.
- comprador, vendedor y vendedorcomprador ya que estos utilizan los protocolos para realizar las transacciones de compraventa con los mercados.
- registro ya que éste utiliza el protocolo de intercambio de reputación para dar información de transacciones de confianza y reputación.

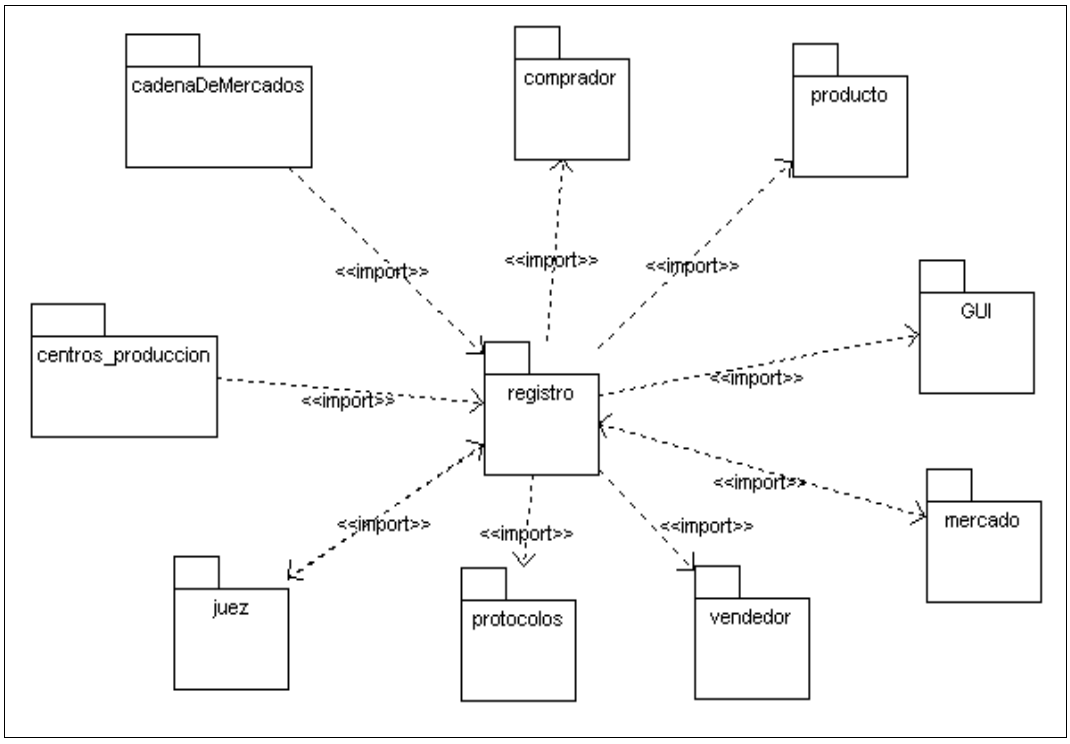


Figura 3.2.1.16: Diagrama de Paquetes.

El paquete registro importa a los siguientes paquetes:

- mercado, comprador, vendedor, producto ya que los utiliza para llevar cuenta de los mercados de la aplicación así como de los compradores y vendedores de cada mercado junto con los productos que van intercambiando en las transacciones que van realizando entre ellos y que quedan reflejadas en el registro.
- GUI para mostrar al usuario por consola información sobre las transacciones que se van realizando y de las que tiene constancia el registro.
- juez para informar del motivo de la penalización a un vendedor.
- protocolos ya que el registro utiliza el protocolo de intercambio de reputación para dar información de transacciones de confianza y reputación.

El paquete registro es importado por los siguientes paquetes:

- cadenaDeMercados ya que aquí es donde se crea el registro único para toda la aplicación.
- centros_produccion para ir añadiendo en el registro las transacciones de generación de productos y los nuevos productos creados.
- juez para ir añadiendo en el registro las transacciones correspondientes al juez y solicitarle datos de los vendedores para evaluar las sanciones.
- mercado para ir añadiendo en el registro las transacciones que se producen en los mercados.

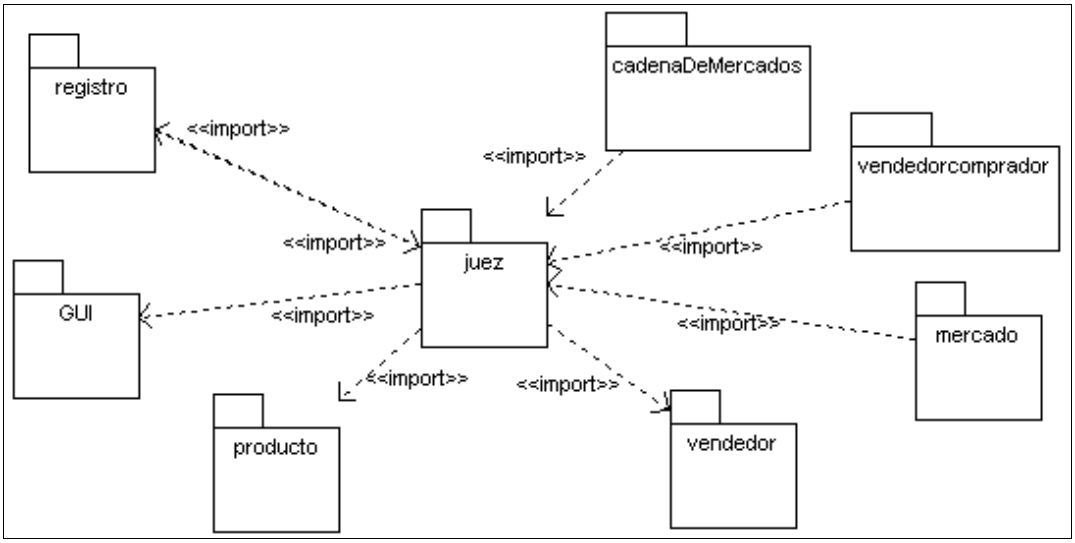


Figura 3.2.1.17: Diagrama de Paquetes.

El paquete juez importa a los siguientes paquetes:

- registro ya que el juez va añadiendo al registro las transacciones correspondientes que va realizando y además le solicita datos de los vendedores para evaluar las sanciones.
- vendedor ya que el juez necesita conocer los vendedores del mercado en el que actúa para poder realizar auditorias y poder sancionarlos si incumplen las reglas.
- producto ya que el juez recibe los productos por los que un comprador ha realizado una queja y comprueba la cantidad y la calidad de los mismos.
- GUI para mostrar al usuario por consola información sobre las acciones realizadas por este.

El paquete juez es importado por los siguientes paquetes:

- cadenaDeMercados ya que en éste es donde se crea el juez que va a regir el mercado de MercaMadrid.
- registro para dar información de las acciones realizadas por el juez.
- mercado ya que uno de los tres mercados contiene un juez para que actúe en caso de que algún vendedor cometa una infracción.
- vendedorcomprador para recibir la notificación de una penalización sufrida.

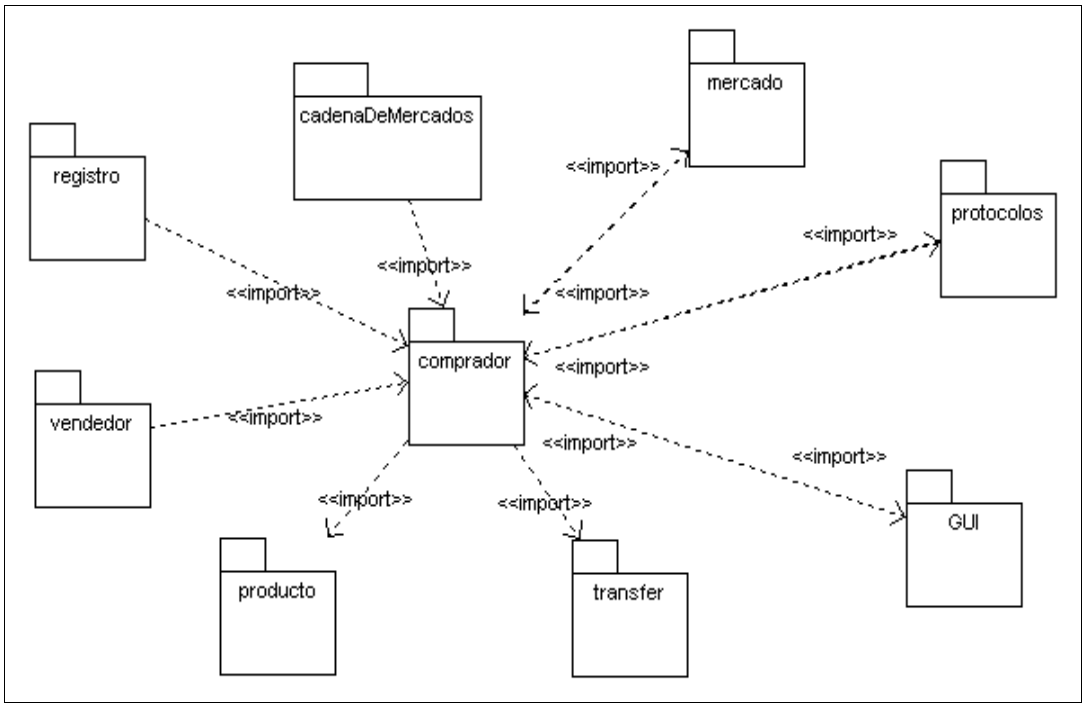


Figura 3.2.1.18: Diagrama de Paquetes.

El paquete comprador importa a los siguientes paquetes:

- mercado ya que los compradores contienen el mercado en el que actúan.
- protocolos ya que los compradores utilizan los protocolos para realizar las transacciones de compraventa con los mercados.
- producto ya que los compradores poseen objetos producto y los van intercambiando con los vendedores en las transacciones.
- GUI para mostrar por consola la información de las transacciones realizadas por los compradores y mostrar en la vista de la aplicación la información de estos.
- transfer ya que los compradores utilizan estos objetos para introducir sus datos y poder pasárselos a la vista para mostrarlos al usuario.

El paquete comprador es importado por los siguientes paquetes:

- cadenaDeMercados ya que aquí es donde se crean los compradores de la aplicación.
- mercado ya que aquí es donde actúan los compradores realizando sus operaciones de compraventa.

- protocolos debido a que éstos implementan métodos que utilizan los compradores y aunque se podrían haber utilizado objetos del tipo Object (estos métodos solo los usan para pasarlos como parámetro de otro método de otro paquete), el ponerlos hace que el código sea más claro.
- registro ya que lleva cuenta de los compradores de cada mercado, así como las transacciones realizadas por ellos.
- vendedor ya que éstos conocen a los compradores con los que realizan operaciones de compraventa.
- GUI ya que utiliza los compradores para mostrar al usuario por medio de la vista de la aplicación, la información de cada uno de éstos.

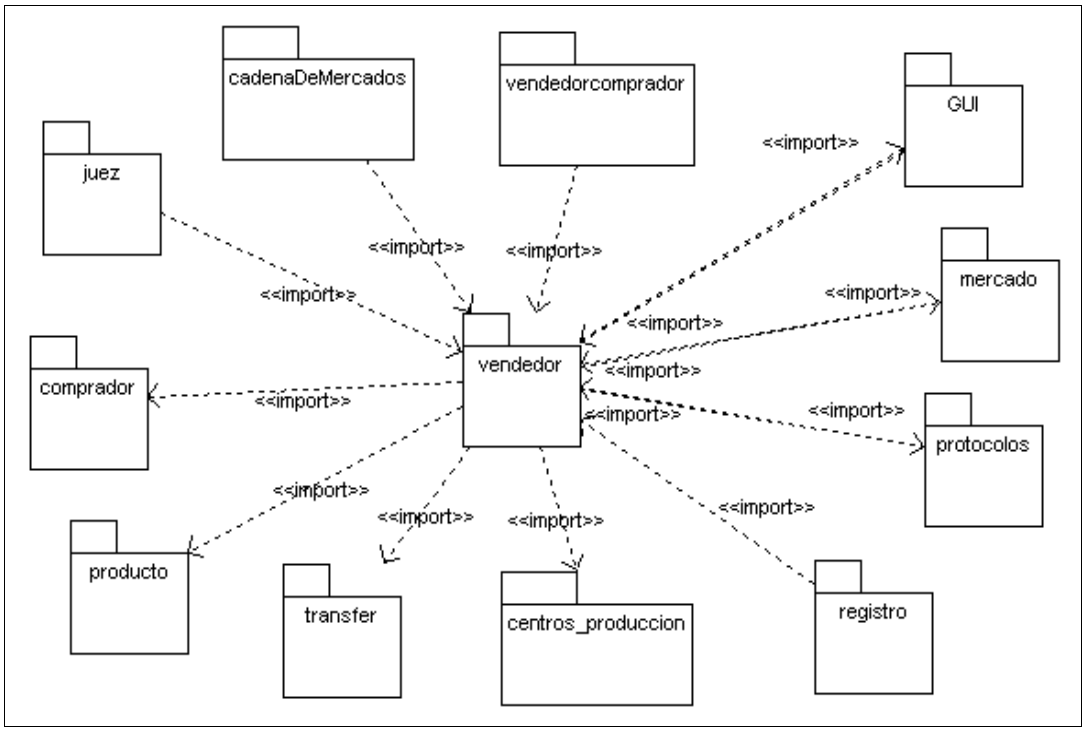


Figura 3.2.1.19: Diagrama de Paquetes.

El paquete vendedor importa a los siguientes paquetes:

- mercado ya que los vendedores contienen el mercado en el que actúan.
- protocolos ya que los vendedores utilizan los protocolos para realizar las transacciones de compraventa con los mercados.
- centros_produccion ya que los vendedores contienen el centro de producción donde fabrican sus productos.

- producto ya que los vendedores poseen objetos producto que solicitan a los centros de producción y los van intercambiando con los compradores en las transacciones.
- comprador ya que los vendedores conocen a los compradores con los que realizan operaciones de compraventa.
- GUI para mostrar por consola la información de las transacciones realizadas por los vendedores y mostrar en la vista de la aplicación la información de estos.
- transfer ya que los vendedores utilizan estos objetos para introducir sus datos y poder pasárselos a la vista para mostrarlos al usuario.

El paquete vendedor es importado por los siguientes paquetes:

- cadenaDeMercados ya que aquí es donde se crean los vendedores de la aplicación.
- mercado ya que aquí es donde actúan los vendedores realizando sus operaciones de compraventa.
- protocolos debido a que éstos implementan métodos que utilizan los vendedores y aunque se podrían haber utilizado objetos del tipo Object (estos métodos solo los usan para pasarlos como parámetro de otro método de otro paquete), el ponerlos hace que el código sea más claro.
- registro ya que lleva la cuenta de los vendedores de cada mercado, así como las transacciones realizadas por ellos.
- vendedorcomprador ya que éstos (intermediarios) conocen a los vendedores con los que realizan operaciones de compraventa.
- GUI ya que utiliza los compradores para mostrar al usuario por medio de la vista de la aplicación, la información de cada uno de éstos.
- juez ya que el éste necesita conocer los vendedores del mercado en el que actúa para poder realizar auditorias y poder sancionarlos si incumplen las reglas.

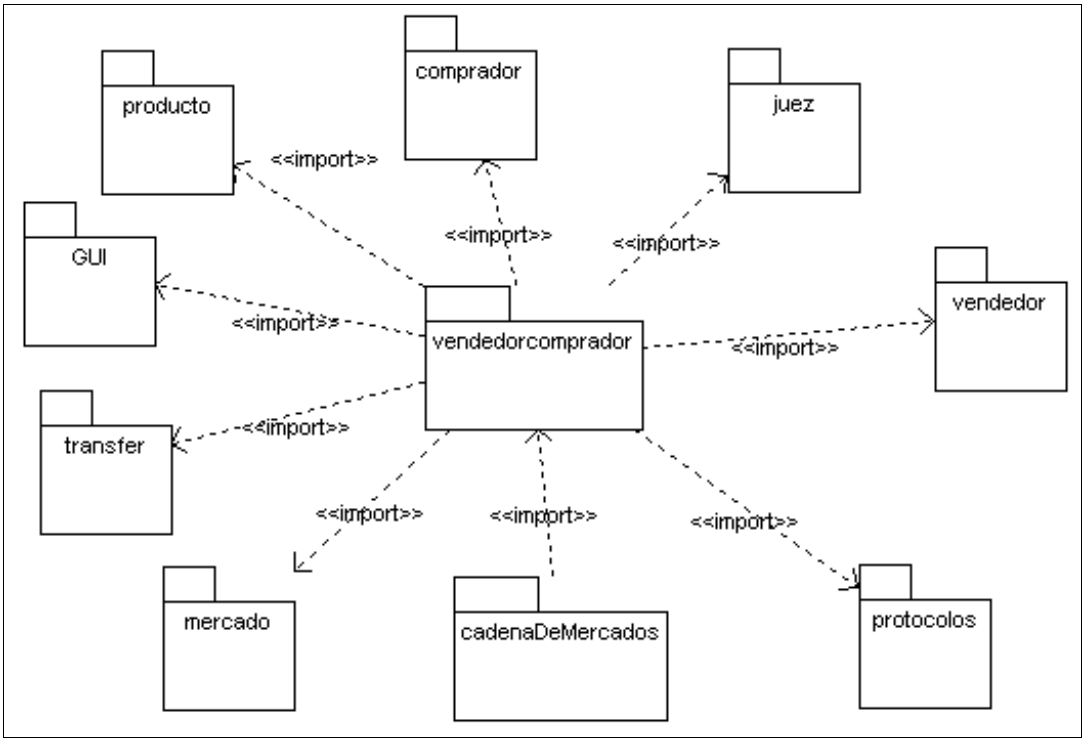


Figura 3.2.1.201: Diagrama de Paquetes.

El paquete vendedorcomprador importa a los siguientes paquetes:

- mercado ya que los vendedores-compradores contienen el mercado en el que actúan.
- protocolos ya que los vendedores-compradores utilizan los protocolos para realizar las transacciones de compraventa con los mercados.
- producto ya que los vendedores-compradores poseen objetos producto y los van intercambiando con los compradores y con los vendedores en las transacciones.
- comprador y vendedores ya que los vendedores-compradores conocen a los compradores y los vendedores con los que realizan operaciones de compraventa.
- GUI para mostrar por consola la información de las transacciones realizadas por los vendedores-compradores y mostrar en la vista de la aplicación la información de estos.
- transfer ya que los vendedores-compradores utilizan estos objetos para introducir sus datos y poder pasárselos a la vista para mostrarlos al usuario.
- juez para recibir la notificación de una penalización sufrida.

El paquete vendedor es importado por los siguientes paquetes:

- cadenaDeMercados ya que aquí es donde se crean los vendedores-compradores de la aplicación.

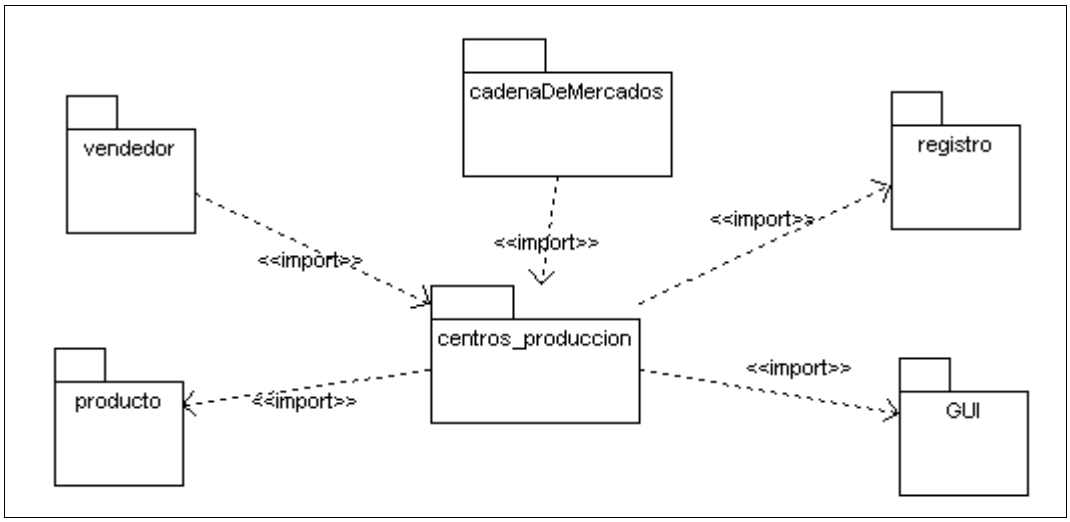


Figura 3.2.1.21: Diagrama de Paquetes.

El paquete centros_produccion importa a los siguientes paquetes:

- producto ya que en los centros de producción es donde se crean los objetos producto de la aplicación.
- registro para pasarle a los centros de producción el registro general de la aplicación e ir añadiendo en el las transacciones de generación de productos y los nuevos productos creados.
- GUI para mostrar al usuario por consola información sobre la creación de nuevos productos.

El paquete centros_produccion es importado por los siguientes paquetes:

- cadenaDeMercados ya que aquí se crean los centros de producción de la aplicación, en este caso, una granja.
- vendedor porque estos poseen un centro de producción, en este caso una granja, donde los vendedores generan sus productos.

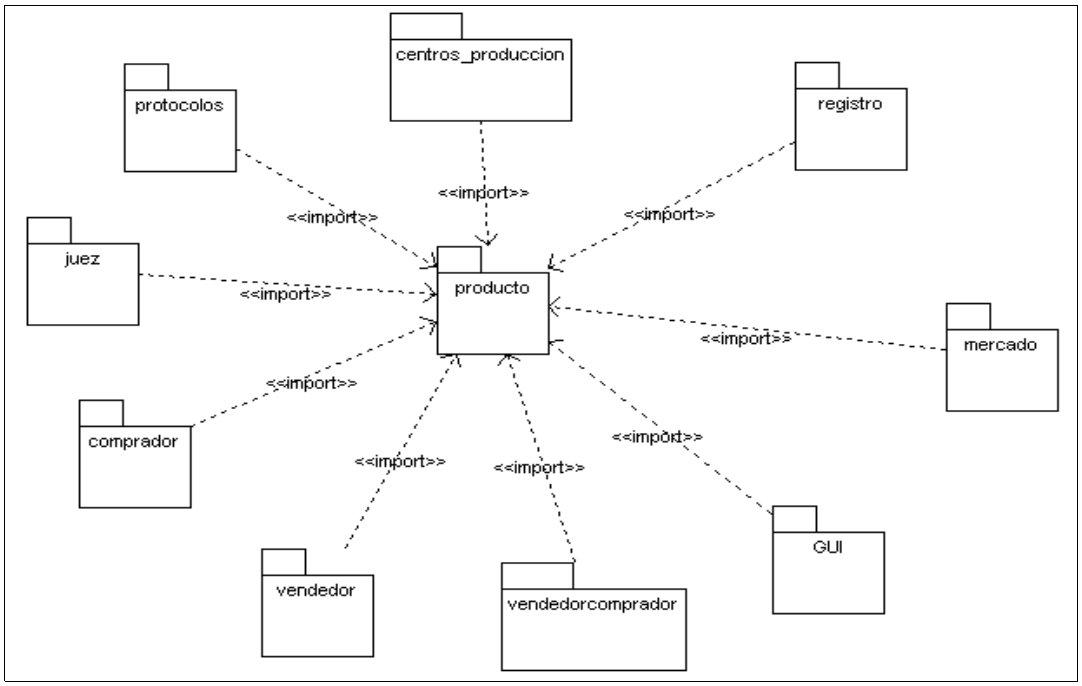


Figura 3.2.1.22: Diagrama de Paquetes.

El paquete producto es importado por los siguientes paquetes:

- centros_produccion ya que en éste es donde se crean los objetos producto de la aplicación.
- comprador, vendedor y vendedorcomprador porque son los que poseen objetos producto y se los van intercambiando entre ellos durante las transacciones.
- juez ya que recibe los productos por los que un comprador ha realizado una queja y comprueba la cantidad y la calidad de los mismos.
- protocolos y mercado ya que en estos se implementan métodos que los utilizan, y aunque se podrían haber utilizado objetos del tipo Object (estos métodos solo los usan para pasarlos como parámetro de otro método de otro paquete), el ponerlos hace que el código sea más claro.
- registro para dar información de los productos que se ponen en venta en los mercados.
- GUI para mostrar al usuario los productos que tiene cada agente.

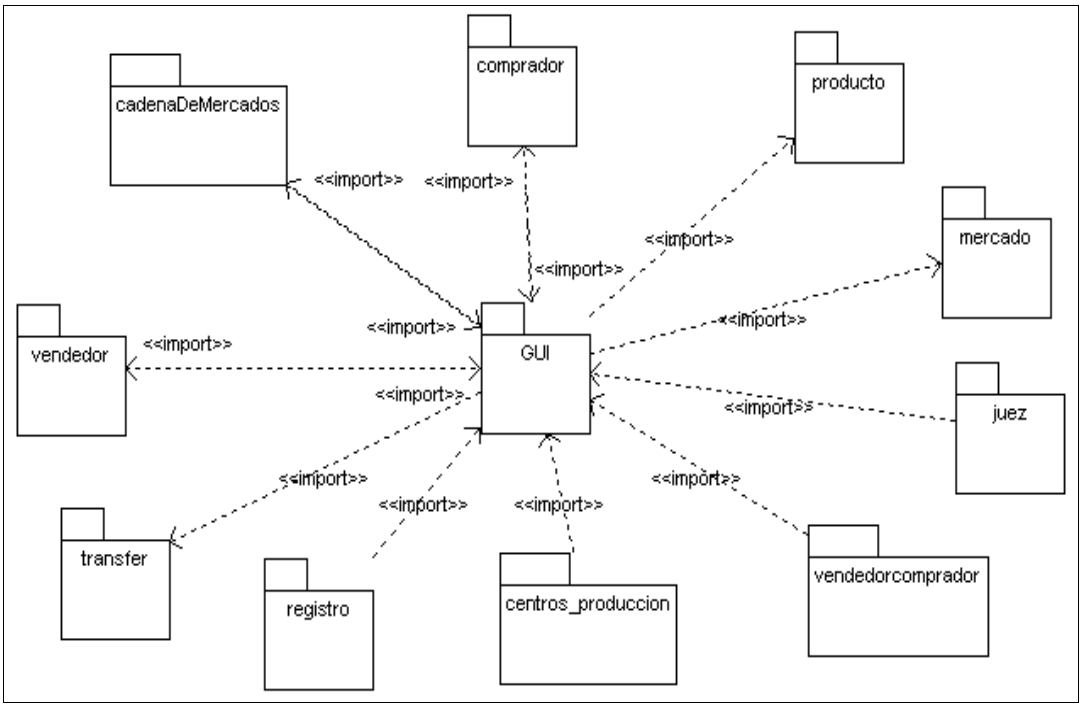


Figura 3.2.1.23: Diagrama de Paquetes.

El paquete GUI importa a los siguientes paquetes:

- cadenaDeMercados ya que en la GUI se crea el modelo de la aplicación (patrón Modelo-Delegado).
- mercado, comprador, vendedor, producto ya que los utiliza para mostrar al usuario por medio de la vista de la aplicación, la información de cada uno de los tres mercados, de los compradores y vendedores de éstos, y de los productos que manejan.
- transfer ya que por medio de estos objetos la vista obtiene la información de los compradores y vendedores para mostrarla al usuario.

El paquete GUI es importado por los siguientes paquetes:

- cadenaDeMercados ya que contiene la vista principal de la aplicación y crea el gestor de gráficas para la generación de las diferentes gráficas con los resultados.
- centros_produccion para mostrar al usuario por consola información sobre la creación de nuevos productos.
- juez para mostrar al usuario por consola información sobre las acciones realizadas por este.

- registro para mostrar al usuario por consola información sobre las transacciones que se van realizando.
- comprador, vendedor y vendedorcomprador para mostrar por consola la información de las transacciones realizadas por estos y para rellenar los transfers para mostrar en la vista de la aplicación la información de estos.

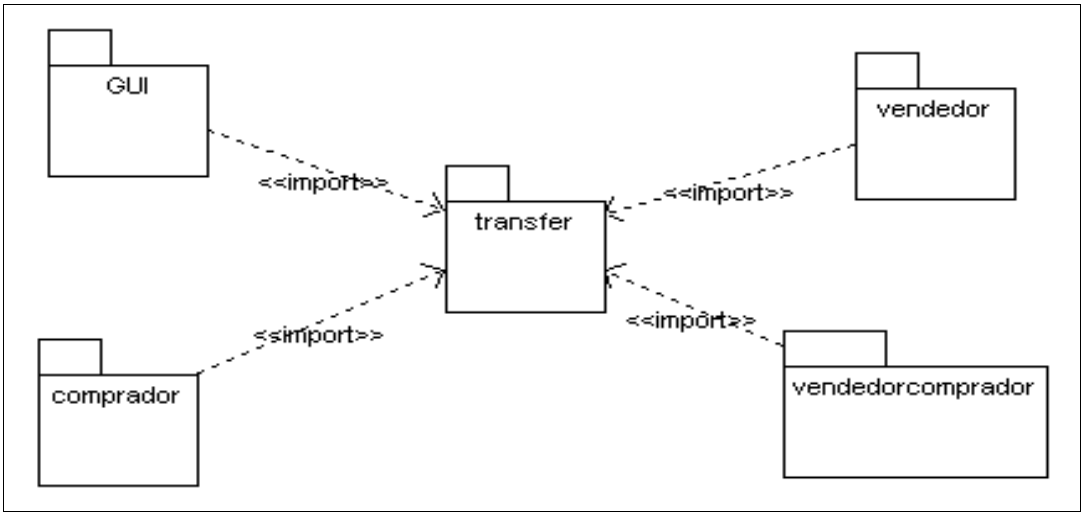


Figura 3.2.1.24: Diagrama de Paquetes.

El paquete transfer es importado por los siguientes paquetes:

- GUI ya que a través de los objetos transfer la vista obtiene la información de los compradores y vendedores para mostrarla al usuario.
- comprador, vendedor y vendedorcomprador ya que todos éstos utilizan objetos transfer para introducir sus datos y poder pasárselos a la vista de la aplicación para mostrarlos al usuario.

3.2.2 Diagramas de Estados:

Con los siguientes dos diagramas se representan los estados por los que pasa tanto un comprador que utiliza el protocolo Contract Net como el que utiliza el protocolo Subasta a la hora de realizar una transacción de compra.

- ***DIAGRAMA COMPRADOR POR PROTOCOLO CONTRACT NET***

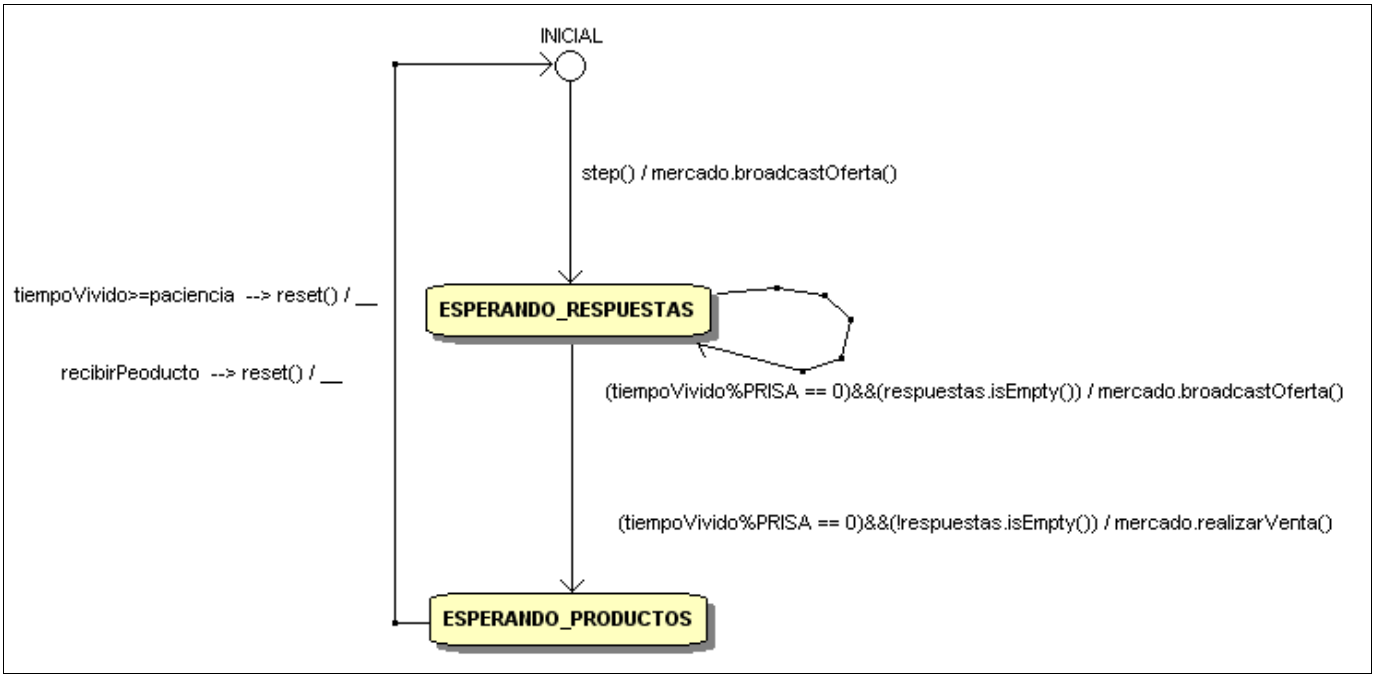


Figura 3.2.2.1: Diagrama de Estados Comprador por el protocolo Contract Net.

El comprador se encuentra al principio en el estado Inicial. Cuando tiene la necesidad de comprar un producto, lo solicita haciendo la petición a los vendedores del mercado. Una vez hace la oferta pasa al estado Esperando_Respuestas. En este estado, si se paso su tiempo de espera y no recibió ninguna propuesta de venta vuelve a realizar otra vez la petición del producto. Si cuando se termina su tiempo de espera recibió alguna propuesta, elige la mejor para sus intereses y le comunica al vendedor que le hizo la oferta que la acepta con lo que pasa al estado Esperando_Productos. Cuando el comprador recibe los productos pactados elimina las demás propuestas de venta que rechazó y pasa al estado Inicial para poder seguir comprando.

- ***DIAGRAMA COMPRADOR POR PROTOCOLO SUBASTA***

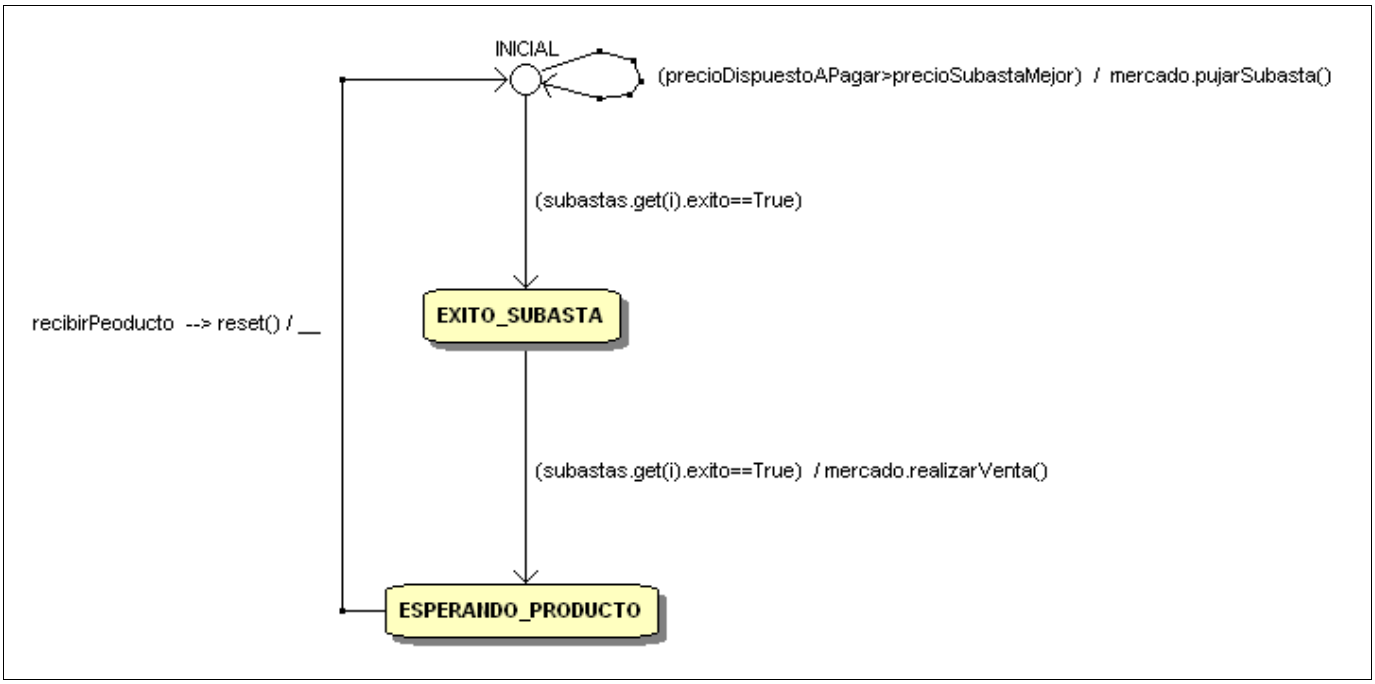


Figura 3.2.2.2: Diagrama de Estados Comprador por el protocolo Subasta.

El comprador se encuentra al principio en el estado Inicial. En este estado el comprador va recibiendo información de las subastas que van creando los vendedores. Con todas las subastas analiza si tienen el producto que necesita y si el precio que tiene el producto en la subasta está dentro de sus posibilidades realiza una puja por el en la subasta. Así, el comprador ira realizando pujas en las diferentes subastas para adquirir los productos que quiere. Cuando se termina una subasta el vendedor informa al comprador que realizó la puja más alta que ha sido el ganador de la subasta. En este momento el comprador pasa al estado Éxito_Subasta. Cuando el comprador está en este estado recorre las subastas que tiene y la que con la que haya obtenido el éxito realiza la venta con dicho vendedor y pasa al estado Esperando_Producto. Cuando el comprador recibe los productos pactados elimina esta subasta y pasa al estado Inicial para poder seguir comprando.

3.2.3.- Diagramas de Casos de Uso:

Las transacciones que realiza la aplicación son las siguientes:

- Transacciones que realizan los Compradores (Consumidores con los vendedores de Mercamadrid y Vendedores de Mercamadrid con los intermediarios): Realizar_Compra_Contract_Net, Solicitar_Confianza, y Solicitar_Reputacion.
- Transacciones que realizan los Consumidores (Compradores de Mercamadird): Realizar_Queja.
- Transacciones que realizan los Productores de Productos: Realizar_Compra_Subasta y Generar_Producto.
- Transacciones que realiza el Juez: Realizar_Auditoria y Penalizar.
- Transacciones que realiza el Usuario: Iniciar_Aplicacion e Iniciar_Simulacion.

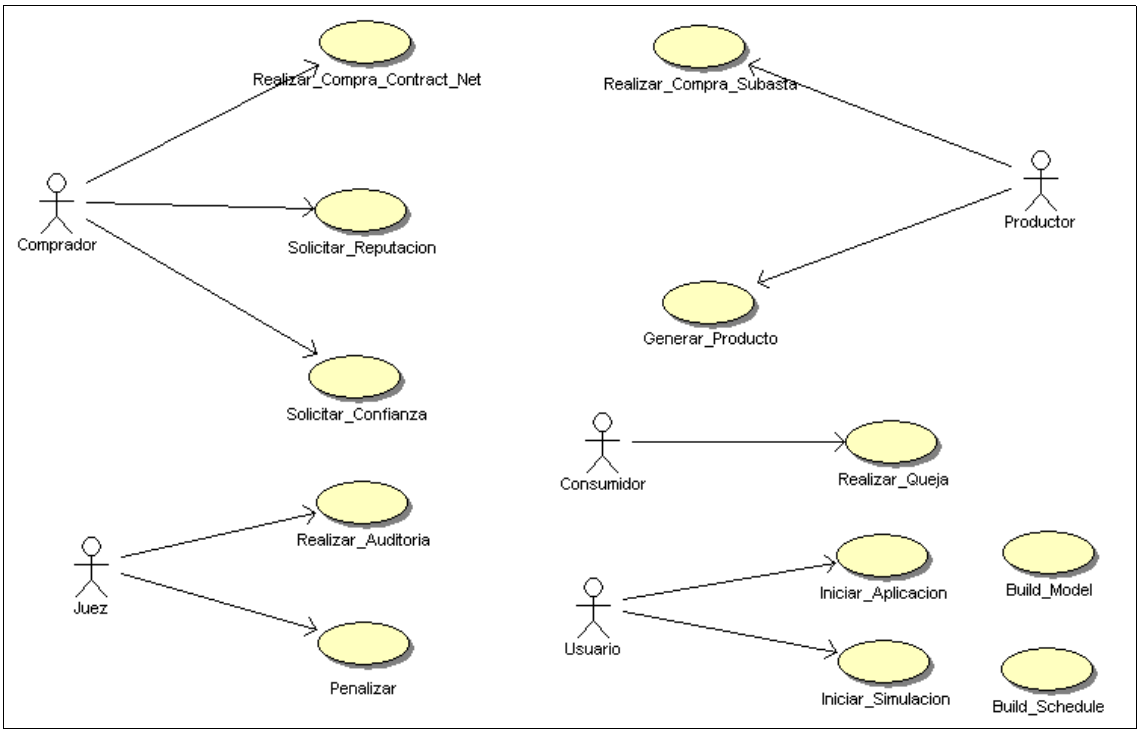


Figura 3.2.3.1: Diagrama de Casos de Uso.

3.2.4.- Diagramas de Secuencia:

Corresponden a las transacciones que se realizan en la aplicación:

- ***DIAGRAMA REALIZAR_COMPRA_CONTRACT_NET***

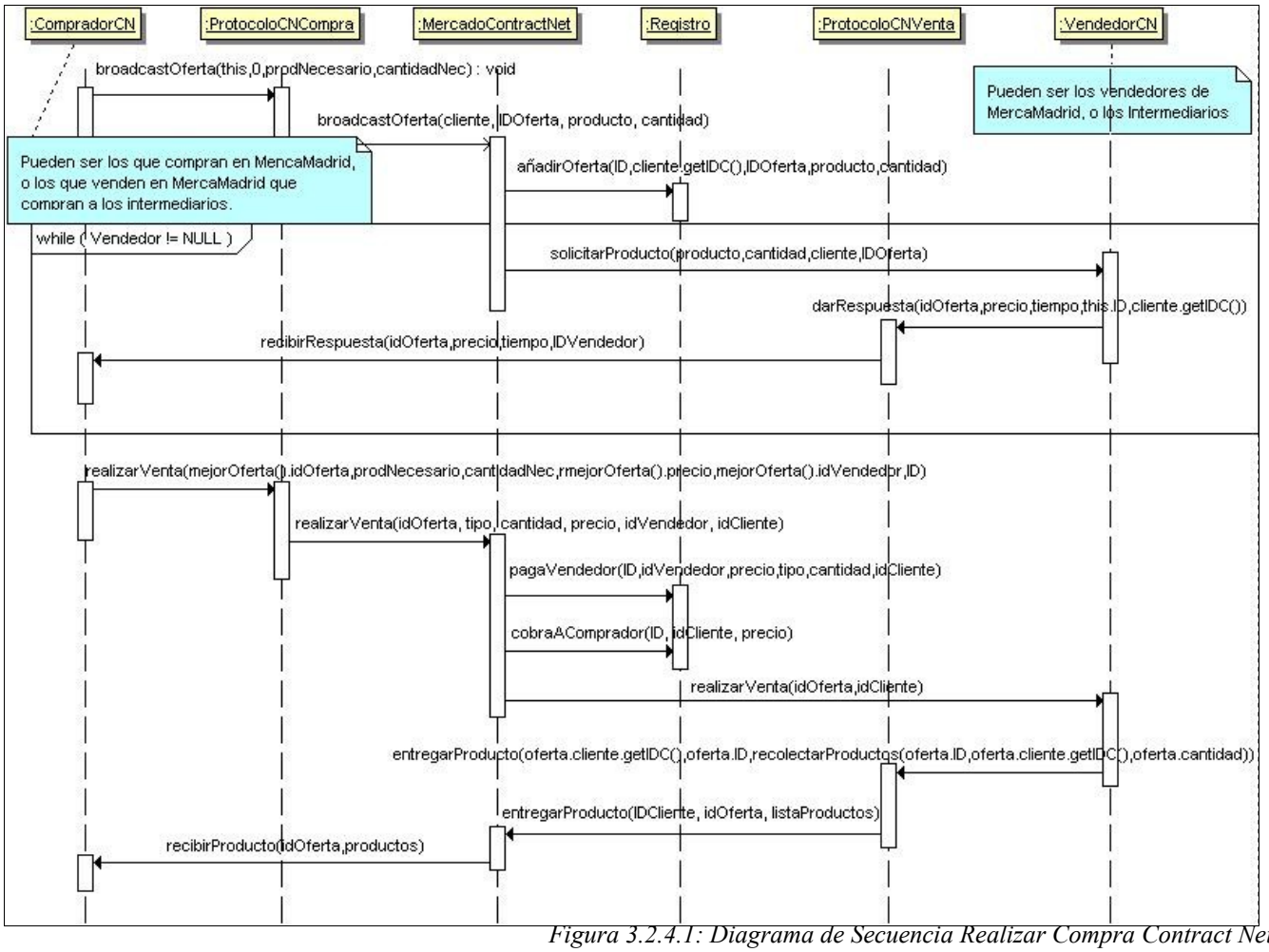


Figura 3.2.4.1: Diagrama de Secuencia Realizar Compra Contract Net.

El comprador del tipo Contract Net (puede ser un cliente que compra en MercaMadrid, o un vendedor de MercaMadrid que se abastece de productos), por medio del protocolo Contract Net Compra realiza un solicitud de un producto al mercado Contract Net. El mercado almacena la solicitud en el registro para que quede constancia y le pasa la solicitud a todos los vendedores Contract Net del mercado (puede ser un Vendedor de MercaMadrid o un intermediario que abastece al vendedor de MercaMadrid). Cada uno de los vendedores examina la

solicitud de compra, y si le interesa realiza una oferta al comprador a través del protocolo Contract Net venta.
Una vez el comprador ha recibido ofertas, elige la que más le interesa, y le pasa al mercado la oferta que ha aceptado a través del protocolo Contract Net compra. En el mercado el comprador paga el dinero pactado, y el vendedor recibe su dinero quedando registrado en el registro. El vendedor una vez realizado esto, recolecta los productos que han pactado, y se los envía al comprador a través del protocolo Contract Net Venta y del mercado. Una vez que el comprador tiene los productos la transacción ha terminado.

- ***DIAGRAMA SOLICITAR_CONFIANZA***

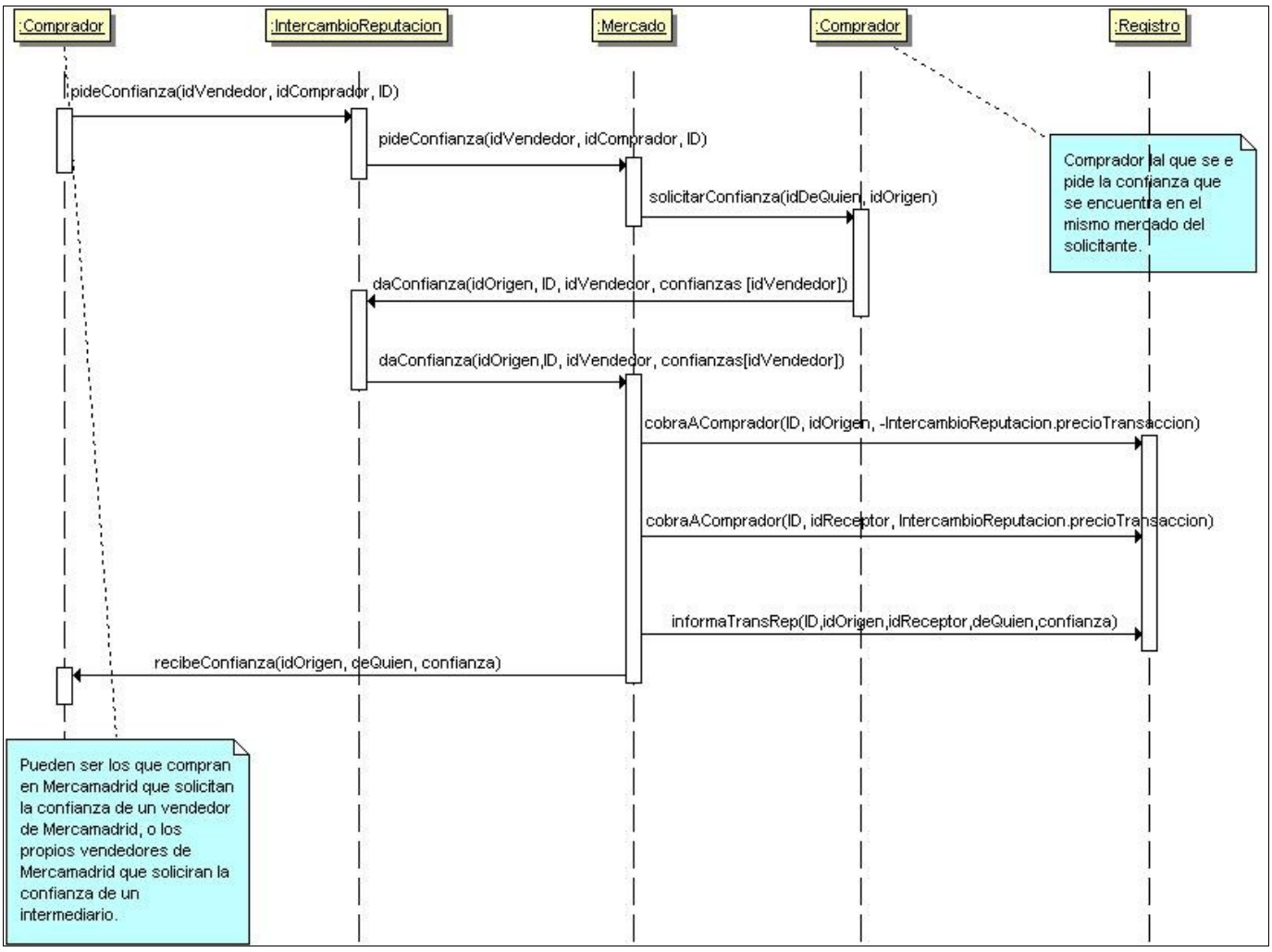


Figura 3.2.4.2: Diagrama de Secuencia Solicitar Confianza.

El comprador (puede ser un cliente que compra en MercaMadrid, o un vendedor de MercaMadrid que se abastece de productos), por medio del protocolo intercambio de reputación le pasa al mercado que quiere pedirle a otro comprador, la confianza que tiene sobre un vendedor. El mercado busca a ese comprador y le pasa la solicitud. Este entrega la confianza que tiene sobre el vendedor solicitado y la pasa al mercado pasando por el protocolo intercambio de reputación. Una vez allí, se refleja la transacción en el registro, el comprador paga la cantidad predefinida por los intercambios de confianza y el comprador al que se le ha pedido recibe el dinero. Una vez esto el comprador solicitante recibe la confianza y termina la transacción.

- ***DIAGRAMA SOLICITAR_REPUTACIÓN***

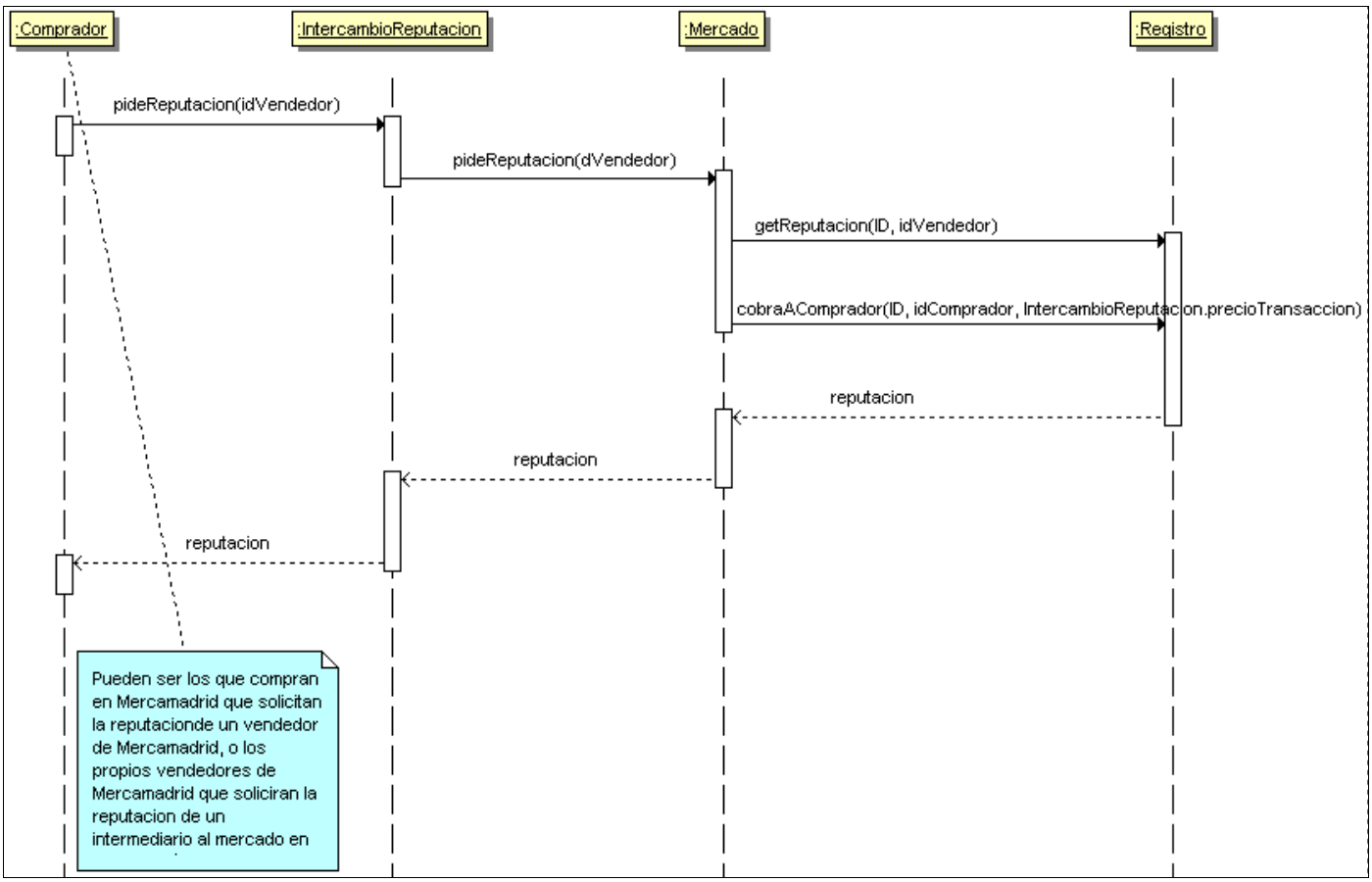


Figura 3.2.4.3: Diagrama de Secuencia Solicitar Reputación.

El comprador (puede ser un cliente que compra en MercaMadrid, o un vendedor de MercaMadrid que se abastece de productos), por medio del protocolo intercambio de reputación le pasa al mercado la identidad del vendedor del que quiere conocer su reputación. El mercado busca la reputación que tiene almacenada de ese vendedor en el registro. Una vez encontrada, se refleja la transacción en el registro, y el comprador paga la cantidad predefinida por los intercambios de reputación. Una vez esto el comprador solicitante recibe la reputación a través del mercado y del protocolo intercambio de reputación y termina la transacción.

- ***DIAGRAMA REALIZAR_COMPRA_SUBASTA***

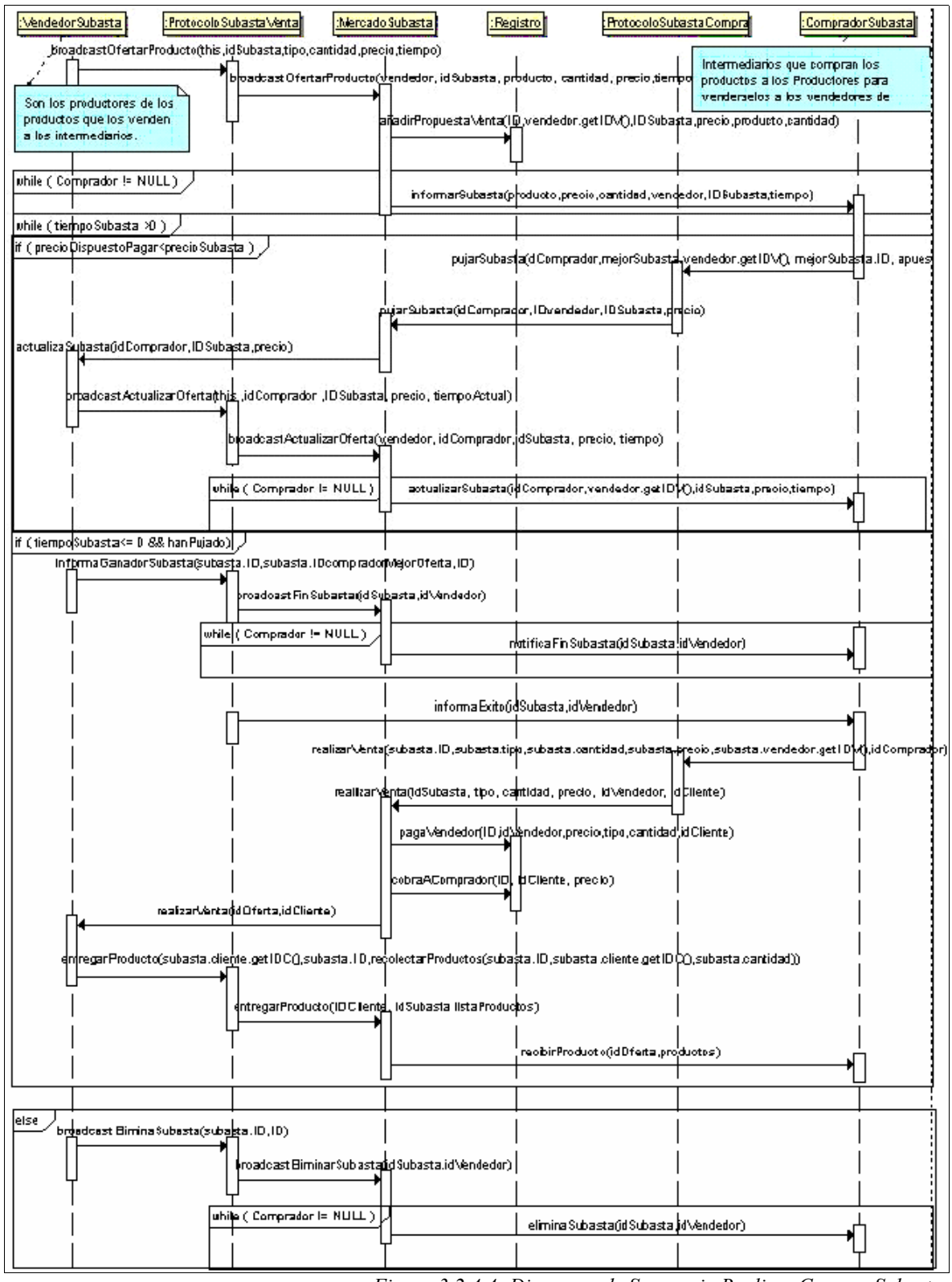


Figura 3.2.4.4: Diagrama de Secuencia Realizar Compra Subasta.

El VendedorSubasta (es uno de los productores de productos) por medio del protocolo Subasta Venta, crea una subasta con alguno de los productos que tiene y se la pasa al mercado Subasta. Este da constancia al registro de la transición que se ha producido y le pasa a todos los compradores del mercado la subasta creada. Estas subastas tienen un tiempo máximo. Mientras no se llegue a ese tiempo máximo cada uno de los compradores evalúa la subasta y si le interesa el producto subastado y el precio hace una puja por ese producto. Esta puja pasa al mercado y del mercado pasa al vendedor que creo la subasta. Este al ver la nueva puja le dice al mercado a través del protocolo subasta venta que actualice la subasta a todos los compradores. Así los compradores deciden si van pujando o no en la subasta hasta que se alcanza el tiempo fin de subasta.

Cuando se termina el tiempo de la subasta, el vendedor le dice al mercado por medio del protocolo subasta venta que la subasta se ha terminado, y le dice al comprador que al final del tiempo de la subasta tenia la mejor puja que acepta su oferta. Cuando el comprador recibe esta notificación le pasa al mercado el trato que se va a realizar por medio del protocolo subasta compra. En el mercado el comprador paga el dinero pactado, y el vendedor recibe su dinero quedando registrado en el registro. El vendedor una vez realizado esto, recolecta los productos que han pactado, y se los envía al comprador a través del protocolo Subasta Venta y del mercado. Una vez que el comprador tiene los productos la transacción ha terminado.

Si una vez llegado al final del tiempo de la subasta ningún comprador ha realizado una puja le pasa al mercado a través del protocolo Subasta Venta que elimine la subasta, y el mercado recorre a todos los compradores y les dice que eliminen la subasta.

- ***DIAGRAMA GENERAR_PRODUCTO***

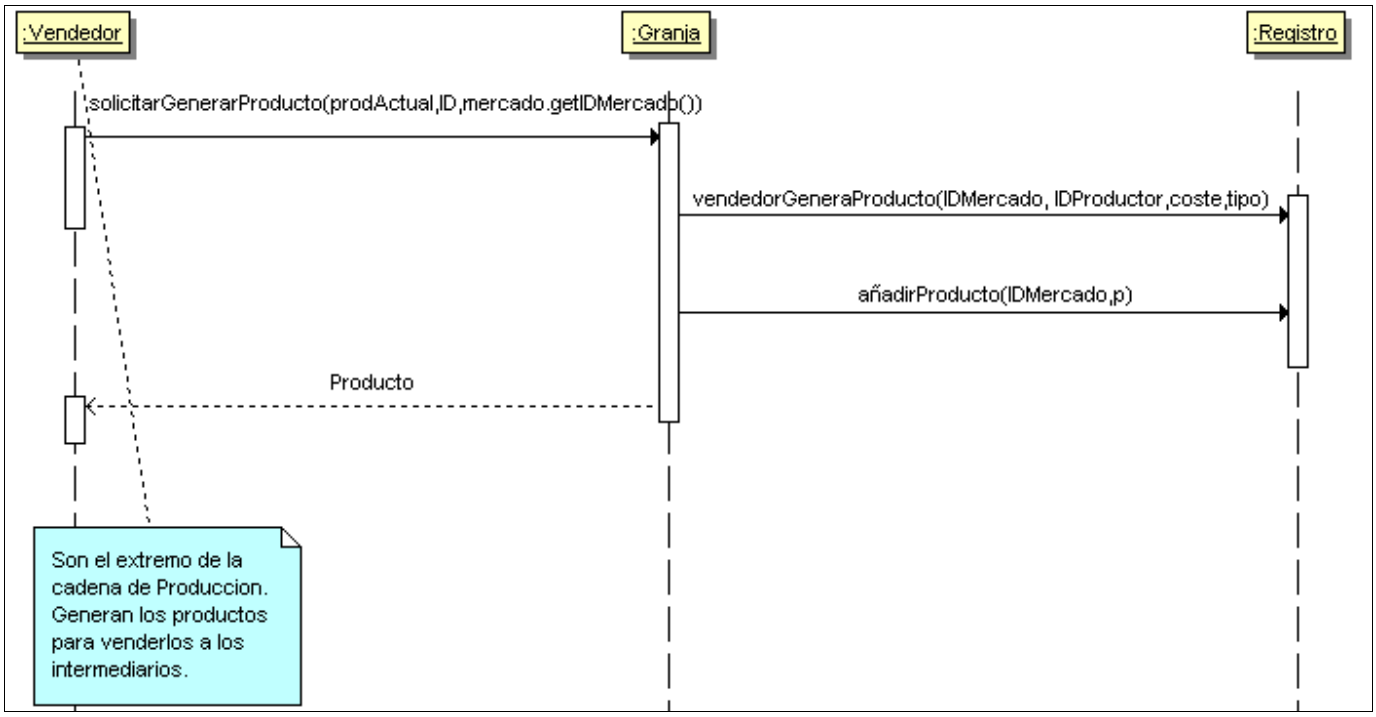


Figura 3.2.4.5: Diagrama de Secuencia Generar Producto.

El Vendedor (es uno de los productores de productos) solicita a la Granja (ahora en la aplicación el único centro de producción implementado es una granja) la creación de un determinado producto. La granja informa al registro de la transacción y de los productos que se van a crear. Una vez hecho esto, la Granja le da al vendedor el producto solicitado.

- ***DIAGRAMA REALIZAR_AUDITORIA***

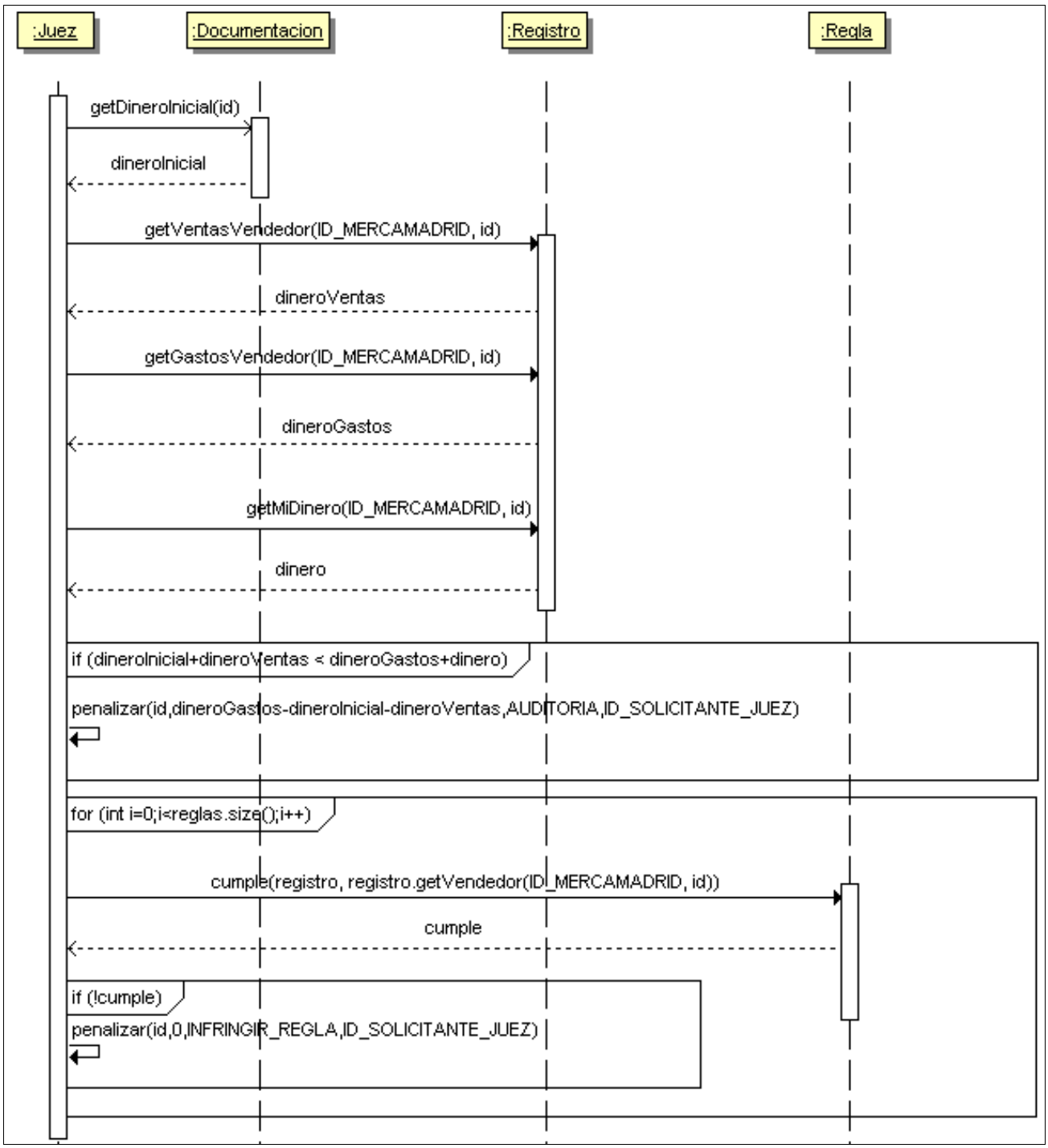


Figura 3.2.4.6: Diagrama de Secuencia Realizar Auditoria.

El Juez, cada cierto tiempo realiza una auditoría a uno de los vendedores de Mercamadrid. Cuando le toca realizarla elige un vendedor al azar y se pone a comprobar sus cuentas. Para ello solicita a la documentación el dinero inicial con el que partió el vendedor, y le solicita al registro de la aplicación el dinero que tiene actualmente, los gastos que ha tenido, y las ventas que realizo. Una vez tiene todos estos datos comprueba que el dinero inicial mas el dinero que gano con sus ventas no sea menor que el dinero que tiene actualmente mas el dinero de los gastos que ha tenido. Si ocurre esto, el vendedor ha cometido una infracción por fraude en las cuentas y es penalizado por una cantidad proporcional a la de la cantidad defraudada. Si por el contrario las cuentas están correctas no ocurre nada. Después de analizar las cuentas, el juez comprueba que el vendedor cumple todas las reglas existentes en el mercado. Si no cumpliera alguna de ellas se le sancionaría de diferente manera dependiendo de la infracción cometida.

- ***DIAGRAMA REALIZAR_QUEJA***

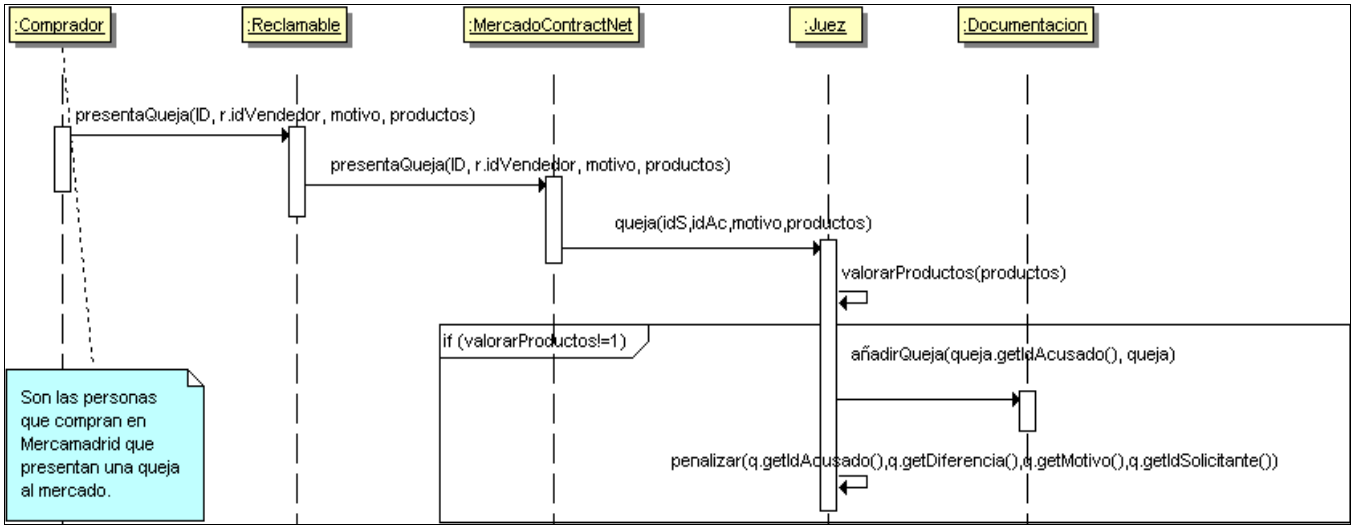


Figura 3.2.4.7: Diagrama de Secuencia Realizar Queja.

El comprador (cliente que compra en MercaMadrid), por medio del protocolo reclamable le pasa al mercado la identidad del vendedor del que quiere presentar una queja por una transacción realizada con él. El mercado le pasa al juez la queja realizada por el comprador y la añade a la documentación para que quede registrada. Después de esto, el juez evalúa la queja y si esta queja tiene justificación el juez penaliza al vendedor dependiendo el nivel de la infracción cometida.

- ***DIAGRAMA PENALIZAR***

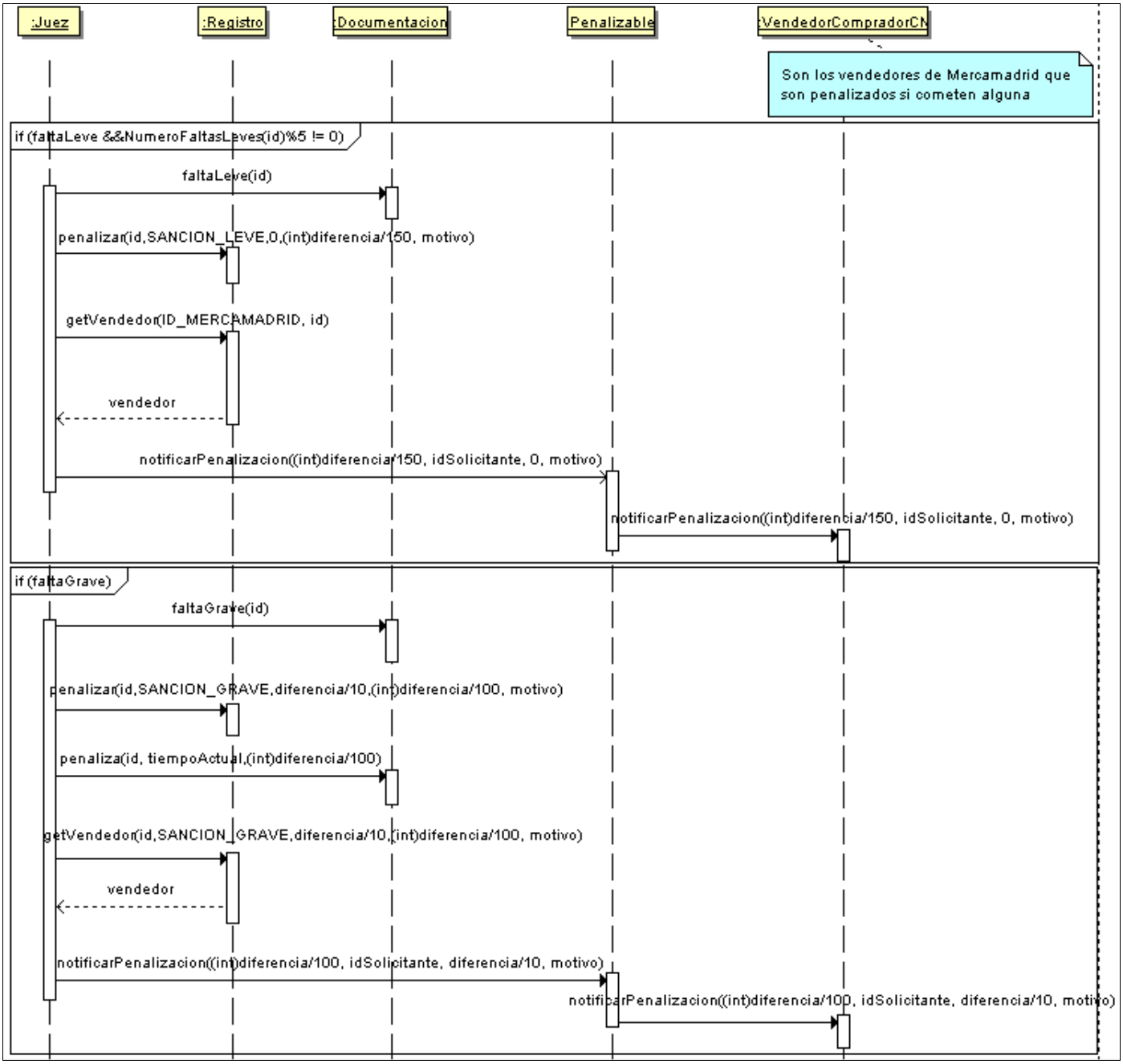


Figura 3.2.4.8: Diagrama de Secuencia Penalizar.

El Juez, cuando se ha cometido una infracción en el mercado penaliza. En este momento, el juez evalúa la infracción cometida.

Si la falta es leve y aun no ha cometido cinco faltas leves, se le penaliza una falta leve. El juez manda a Documentación la falta leve, le pasa al registro toda la información de la sanción para que quede constancia en el, y le notifica la penalización al vendedor que cometió la infracción por medio del protocolo penalizable.
Si la falta es grave o es leve pero es la quinta falta leve, se le penaliza una falta grave. El juez manda a Documentación la falta grave, le pasa al registro toda la información de la sanción para que quede constancia en el, y le notifica la penalización al vendedor que cometió la infracción por medio del protocolo penalizable.

- ***DIAGRAMA INICIAR_APLICACION***

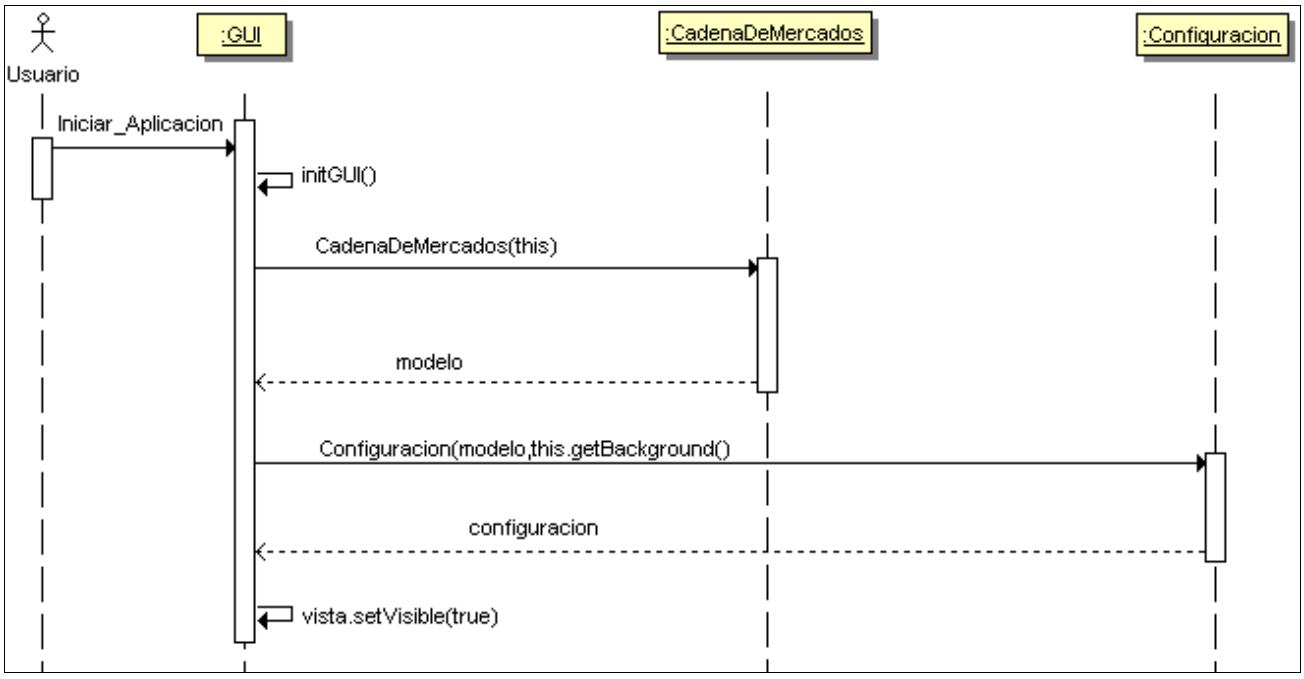


Figura 3.2.4.9: Diagrama de Secuencia Iniciar_Aplicacion.

El usuario inicia la aplicación, se crea la vista principal de la aplicación, después se crea el modelo, se crea la vista para la configuración de los mercados, y para terminar se muestra al usuario la vista principal anteriormente creada.

- ***DIAGRAMA INICIAR_SIMULACION***

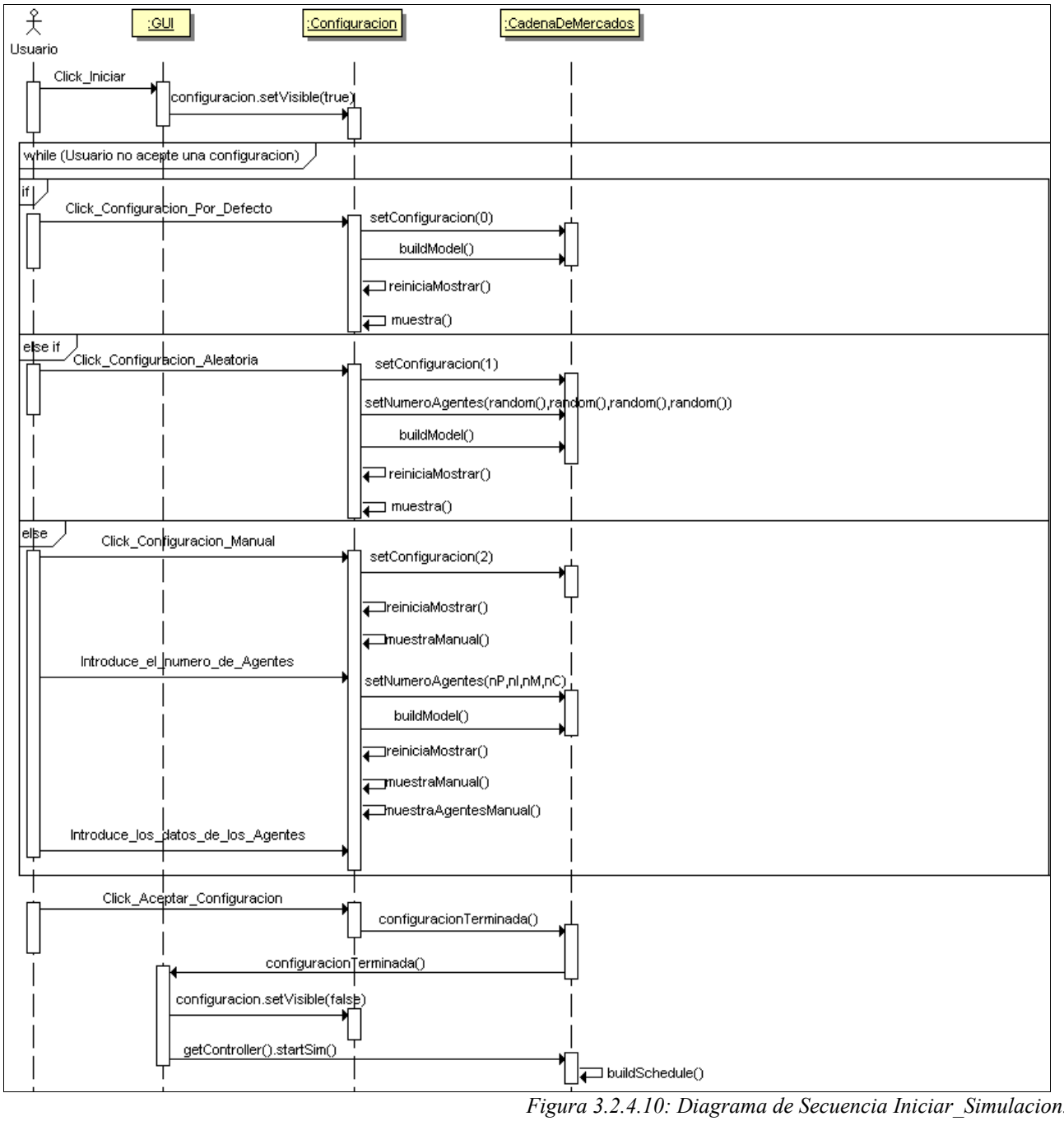


Figura 3.2.4.10: Diagrama de Secuencia Iniciar_Simulacion.

Cuando el usuario hace clic en la opción iniciar de la vista de la aplicación, entonces la GUI hace que la vista de configuración se haga visible al usuario. Una vez que se muestra la configuración al usuario este puede elegir entre las tres

opciones disponibles. Las opciones son configuración por defecto (10 agentes compradores y 10 vendedores por cada uno de los tres mercados), configuración aleatoria (numero de agentes aleatorio en cada uno de los mercados) y configuración manual (el usuario puede elegir el número de agentes por cada mercado además de poder configurar los datos de cada agente). Cada vez que se selecciona una de las tres opciones se llama a buildModel, es decir, construye el modelo (ver diagrama Build_Model) y se muestra en la vista de configuración la opción elegida. Si está de acuerdo pulsa aceptar, si no puede seleccionar otra opción.
Una vez que el usuario ha pulsado el botón aceptar, informa a la vista de que la configuración ha terminado, oculta la vista de configuración y llama a buildSchedule, es decir, programa las acciones a realizar y su periocidad (ver diagrama Build_Schedule).

- ***DIAGRAMA BUILD_MODEL***

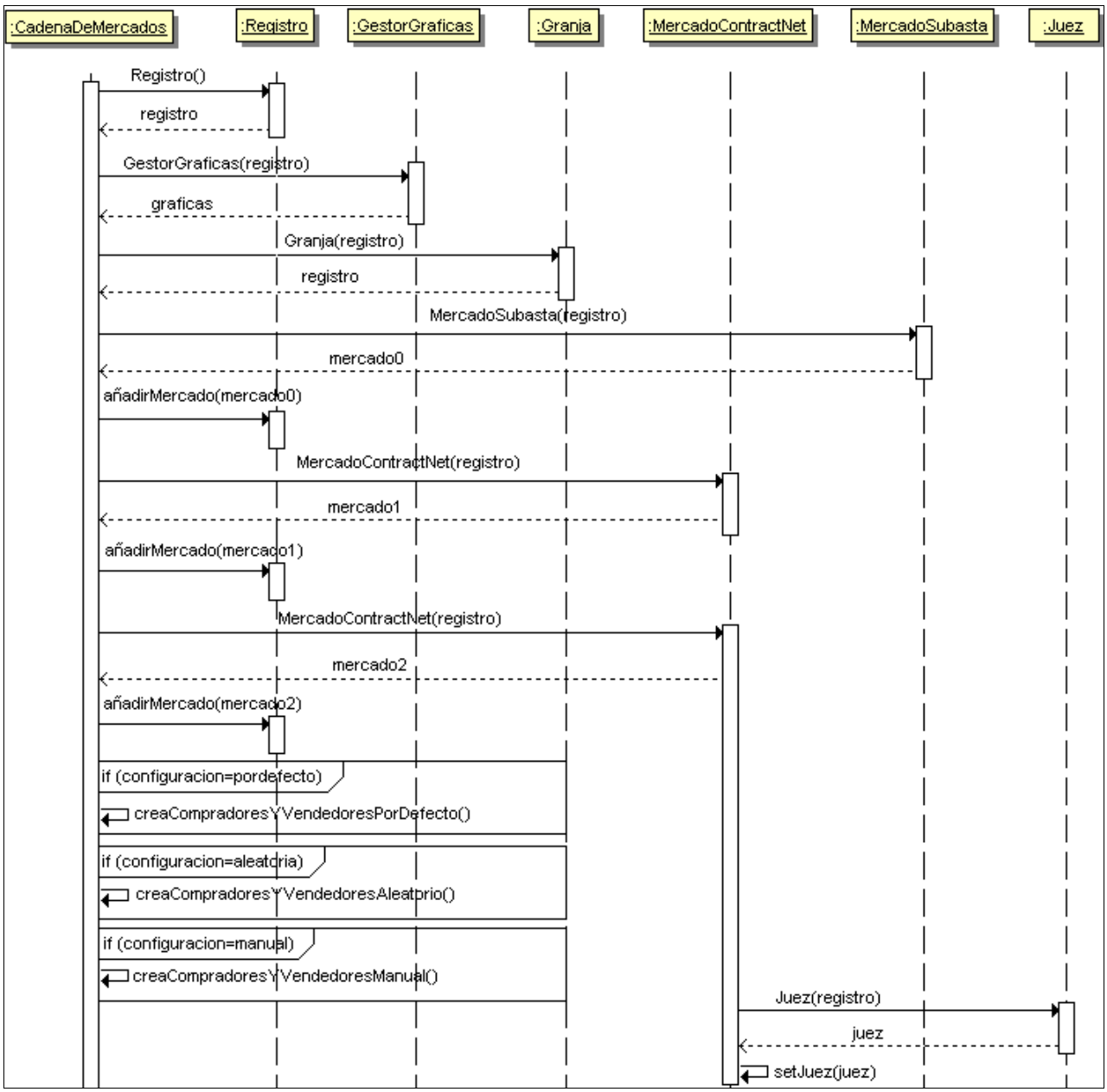


Figura 3.2.4.11: Diagrama de Secuencia Build_Model.

Se crea el registro único para la aplicación, una vez creado este se crea el gestor de gráficas pasándole el registro. Luego se crea la granja a la que también se le pasa el registro. Una vez creado todo esto, lo siguiente es la creación de los mercados a los que también se le pasa el registro de la aplicación. El primer mercado creado es un mercado del tipo subasta (actúan los productores y los intermediarios), el siguiente es un mercado del tipo contract net (actúan los

intermediarios y los vendedores de Mercamadrid) y el último también es del tipo contract net (actúan los consumidores y los vendedores de Mercamadrid). Una vez creados los mercados y añadidos al modelo, dependiendo de la opción elegida por el usuario en la configuración se añade un número u otro de agentes con unas determinadas características en cada mercado. Por último se crea el juez, que actúa en Mercamadrid, pasándole el registro de la aplicación y colocándolo en el mercado de Mercamadrid (último mercado creado).

- ***DIAGRAMA BUILD_SCHEDULE***

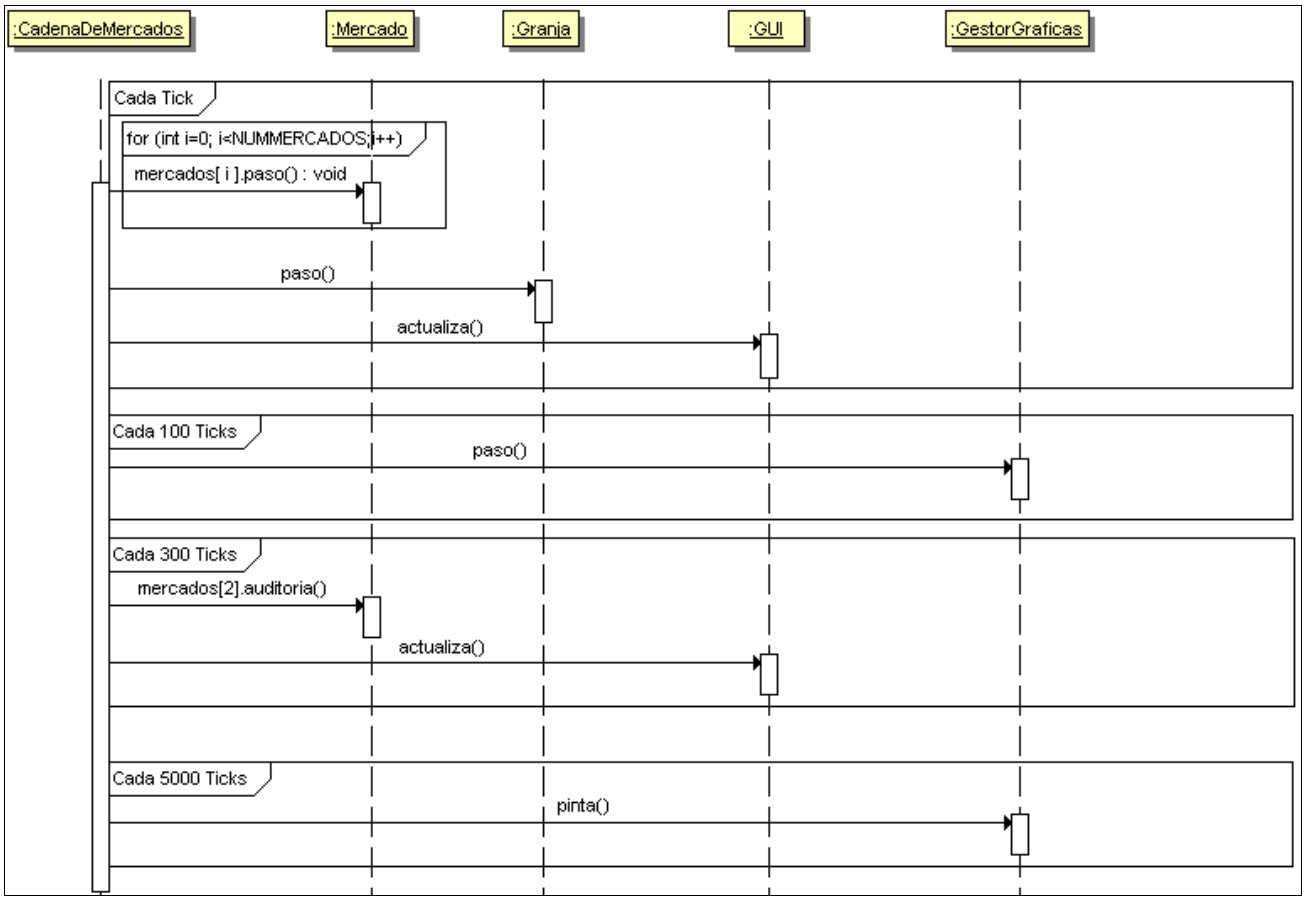


Figura 3.2.4.12: Diagrama de Secuencia Build_Schedule

- Cada tick se llama al método paso de cada uno de los tres mercados y al método paso de la granja para que se realice la simulación de los agentes; y al método actualizar de la vista de la aplicación.

- Cada 100 ticks se llama al método paso del gestor de gráficas para tomar datos para poder hacer las gráficas para mostrárselas al usuario por la vista de la aplicación.
- Cada 300 ticks se le dice al mercado que hay que realizar una auditoría, entonces este le dice al juez que realice una auditoría a un vendedor al azar y se le dice a la vista que se actualice.
- Cada 5000 ticks se llama al gestor de gráficas con el método pinta para que dibuje las gráficas de resultados en la vista de la aplicación.

3.2.5.- Diagramas de Actividades:

Representan las transacciones que se realizan en la aplicación.

- ***DIAGRAMA REALIZAR_COMPRA_CONTRACT_NET***

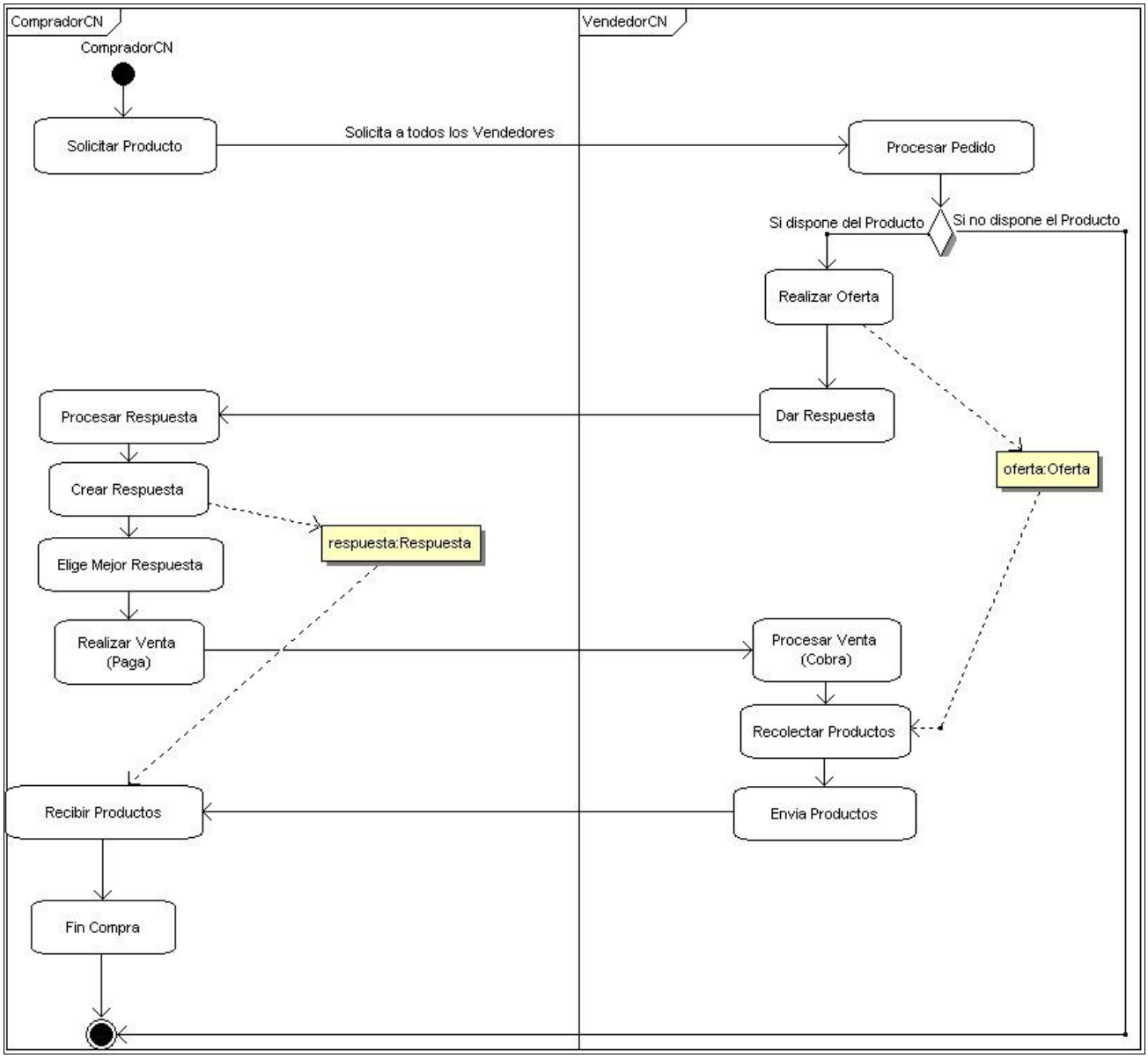


Figura 3.2.5.1: Diagrama de Actividades Realizar Compra Contract Net.

Se representan las acciones que realizan el comprador Contract Net (cliente que compra en MercaMadrid, o vendedor de MercaMadrid que se abastece de productos) y el vendedor Contract Net (Vendedor de MercaMadrid o

intermediario que abastece al vendedor de MercaMadrid) para realizar una acción de compraventa mediante el protocolo Contract Net.

El comprador solicita un producto, el vendedor le hace una propuesta de venta (se crea un objeto de la clase Oferta en el vendedor, y un objeto de la clase Respuesta en el comprador), el comprador acepta una de las ofertas recibidas, paga la cantidad pactada y el vendedor le entrega los productos al comprador y se termina la transacción.

- ***DIAGRAMA SOLICITAR_CONFIANZA***

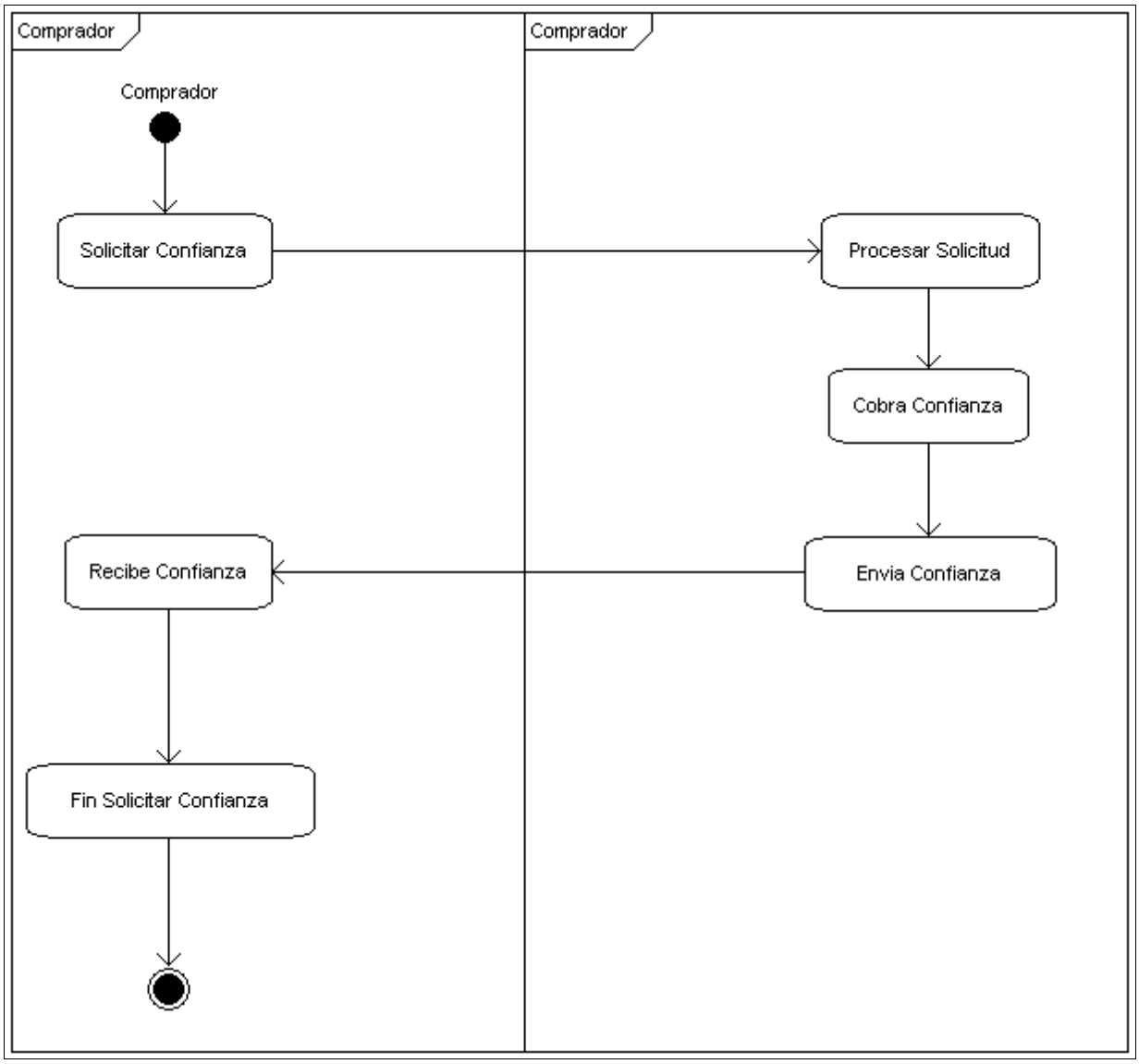


Figura 3.2.5.2: Diagrama de Actividades Solicitar Confianza.

Se representan las acciones que realizan un comprador (cliente que compra en MercaMadrid, o vendedor de MercaMadrid que se abastece de productos) y otro comprador del mismo mercado para la solicitud de confianza de un agente.

El comprador le solicita al otro comprador la confianza de un vendedor (Vendedor de MercaMadrid o intermediario que abastece al vendedor de MercaMadrid), le paga el dinero por la transacción, y recibe del otro comprador la información de la confianza del vendedor solicitado.

- ***DIAGRAMA SOLICITAR_REPUTACION***

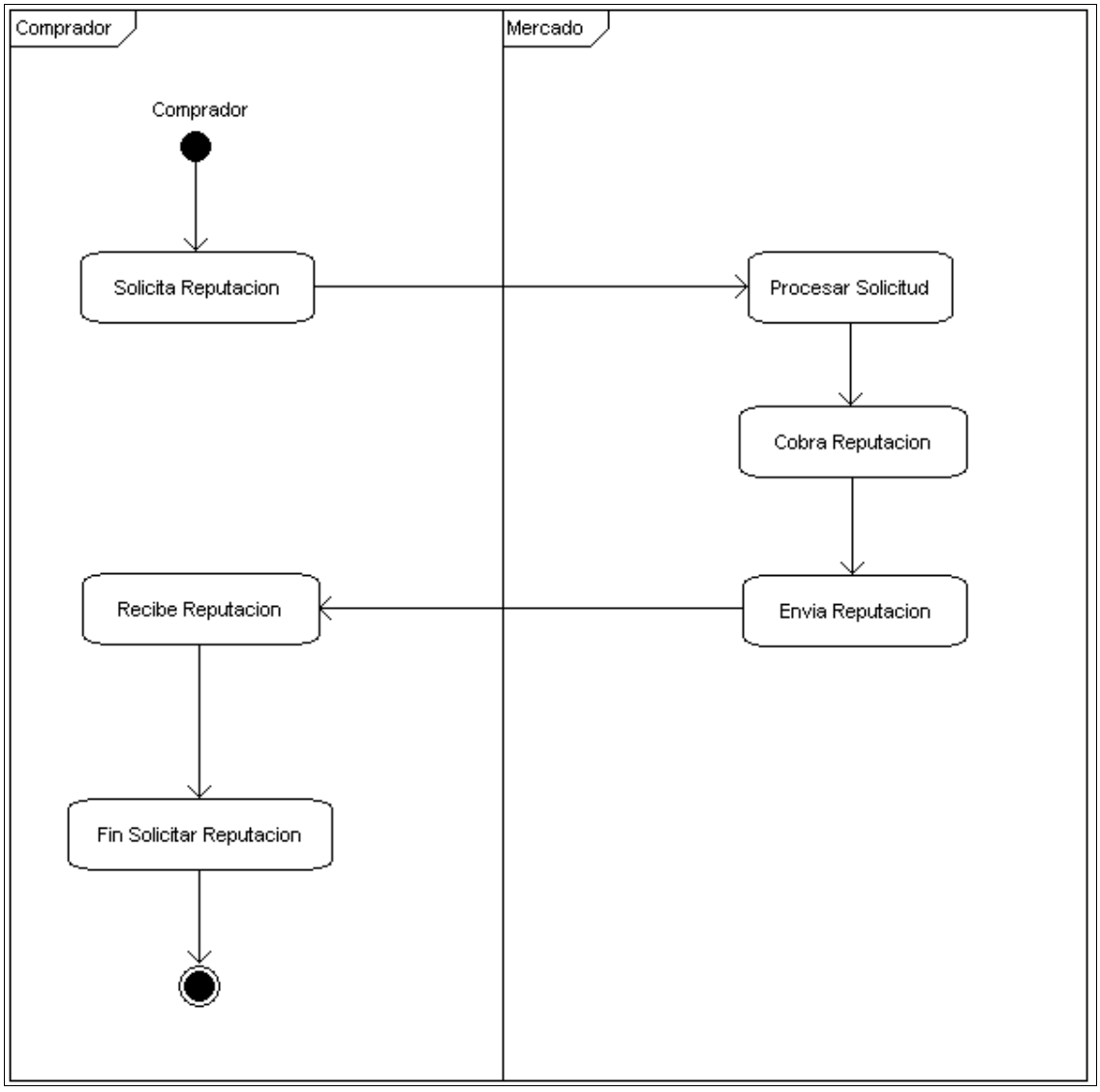


Figura 3.2.5.3: Diagrama de Actividades Solicitar Reputación.

Se representan las acciones que realizan un comprador (cliente que compra en MercaMadrid, o vendedor de MercaMadrid que se abastece de productos) y el mercado para la solicitud de reputación de un agente.

El comprador le solicita al mercado la reputación de un vendedor (Vendedor de MercaMadrid o intermediario que abastece al vendedor de MercaMadrid), le paga el dinero por la transacción, y recibe del mercado la información de la reputación del vendedor solicitado.

- ***DIAGRAMA REALIZAR_COMPRA_SUBASTA***

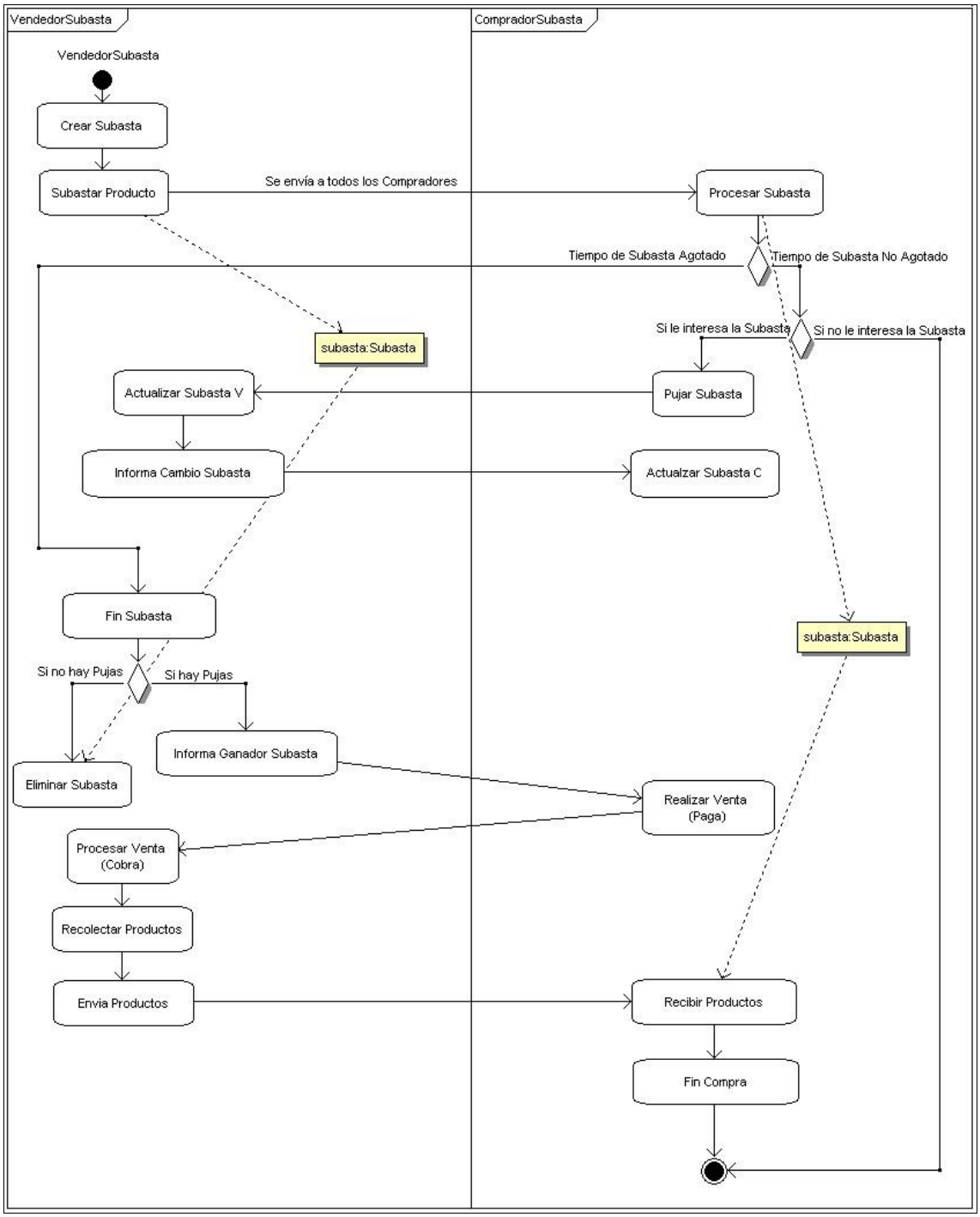


Figura 3.2.5.4: Diagrama de Actividades Realizar Compra Subasta.

Se representan las acciones que realizan el vendedor Subasta (productor de los productos de la cadena de distribución) y el comprador Subasta (intermediario que abastece a los vendedores de MercaMadrid) para realizar una acción de compraventa mediante el protocolo Subasta.

El vendedor pone en subasta un producto (crea un objeto de la clase Subasta del vendedor), los compradores reciben la subasta (crea un objeto de la clase subasta del comprador) pujan o no según les interese la subasta. Cuando llega el fin de la subasta se elimina si no pujó nadie, o se realiza la venta con el comprador que hizo la mejor oferta, este comprador paga los productos y el vendedor le entrega los productos al comprador y se termina la transacción.

- ***DIAGRAMA GENERAR_PRODUCTO***

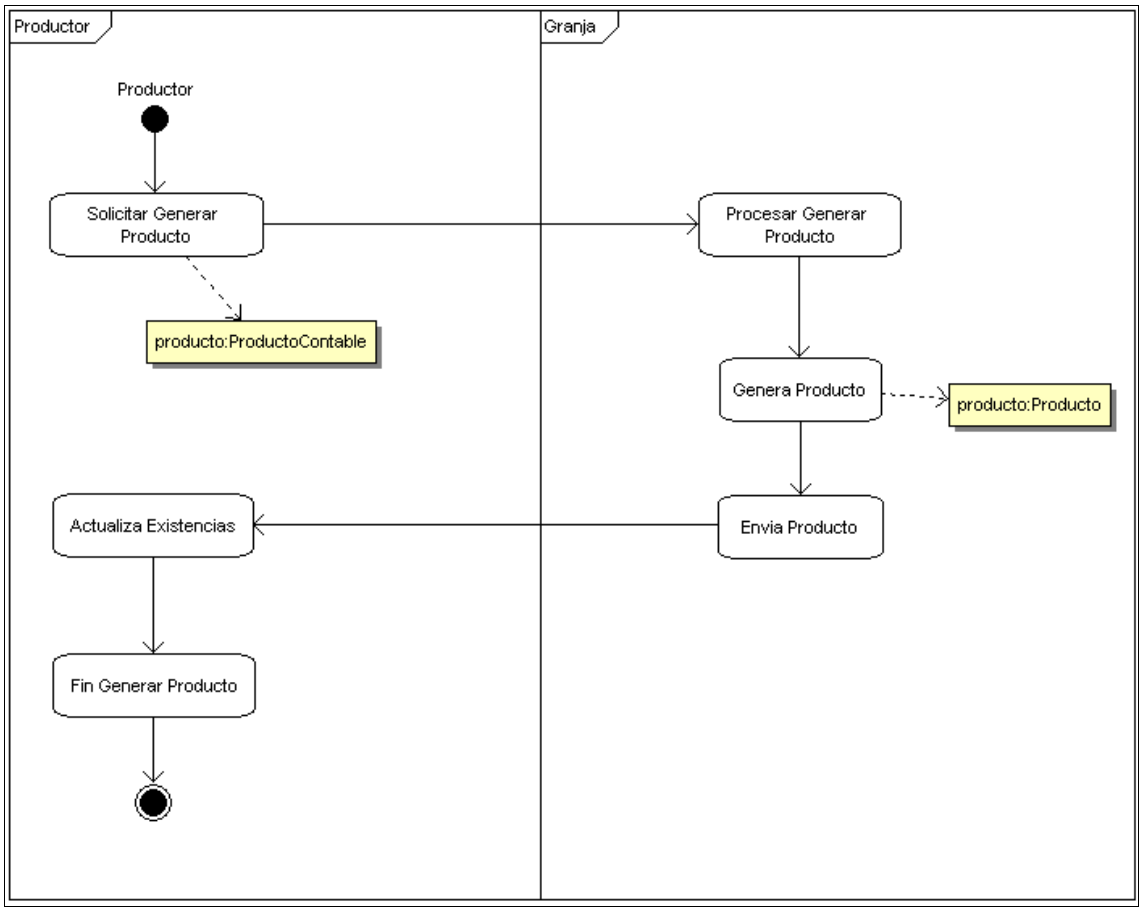


Figura 3.2.5.5: Diagrama de Actividades Generar Producto.

Se representan las acciones que realizan el Productor (agente que crea los productos que se introducen en la cadena de distribución) y la Granja (lugar donde se crean los productos) para la generación de un nuevo producto.

El productor solicita la generación de un producto (se crea un objeto de la clase ProductoContable), entonces en la granja se genera el producto (se crea un objeto de la clase Producto) y el productor recibe el producto creado terminando la transacción.

- ***DIAGRAMA REALIZAR AUDITORIA***

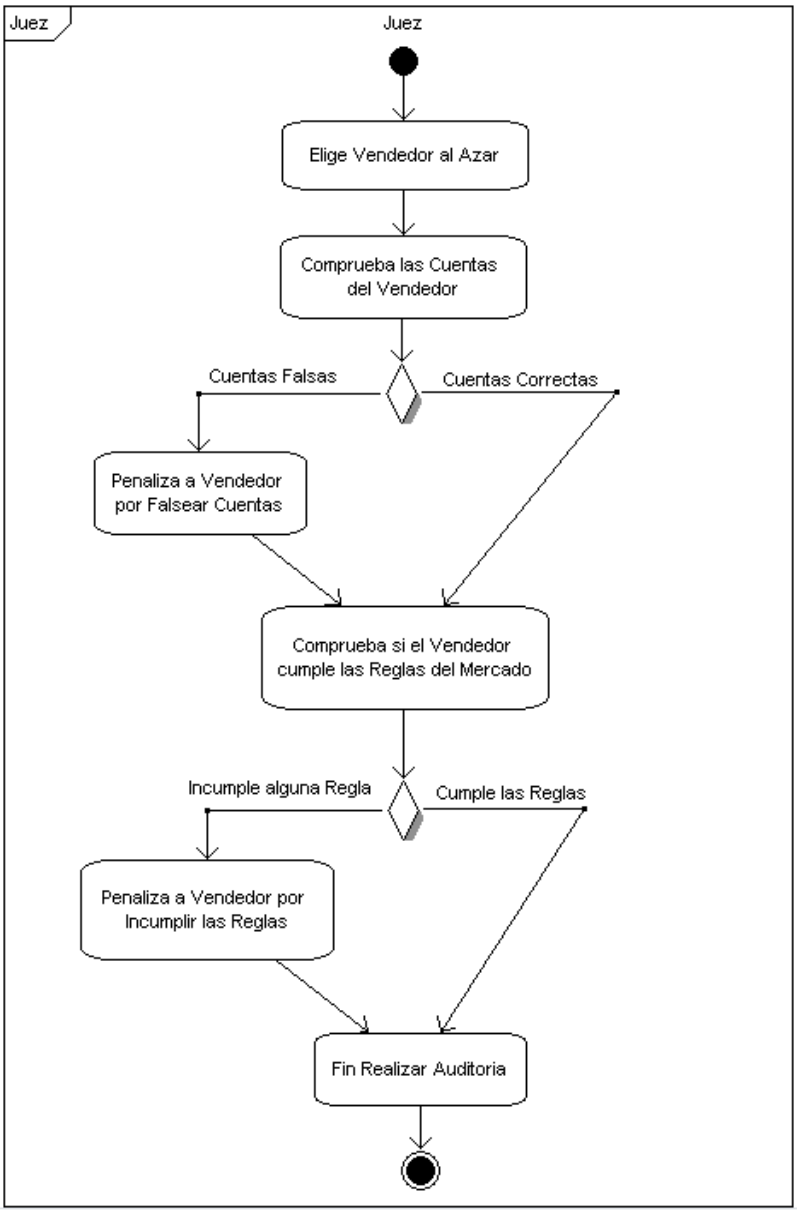


Figura 3.2.5.6: Diagrama de Actividades Realizar Auditoria.

Se representan las acciones que realizan el Juez para realizar una auditoría a un vendedor de Mercamadrid.

El juez elige un vendedor al azar, comprueba sus cuentas y lo penaliza si estas no están bien; y comprueba que cumpla las reglas del mercado y lo sanciona si el vendedor incumple alguna de estas.

- ***DIAGRAMA REALIZAR QUEJA***

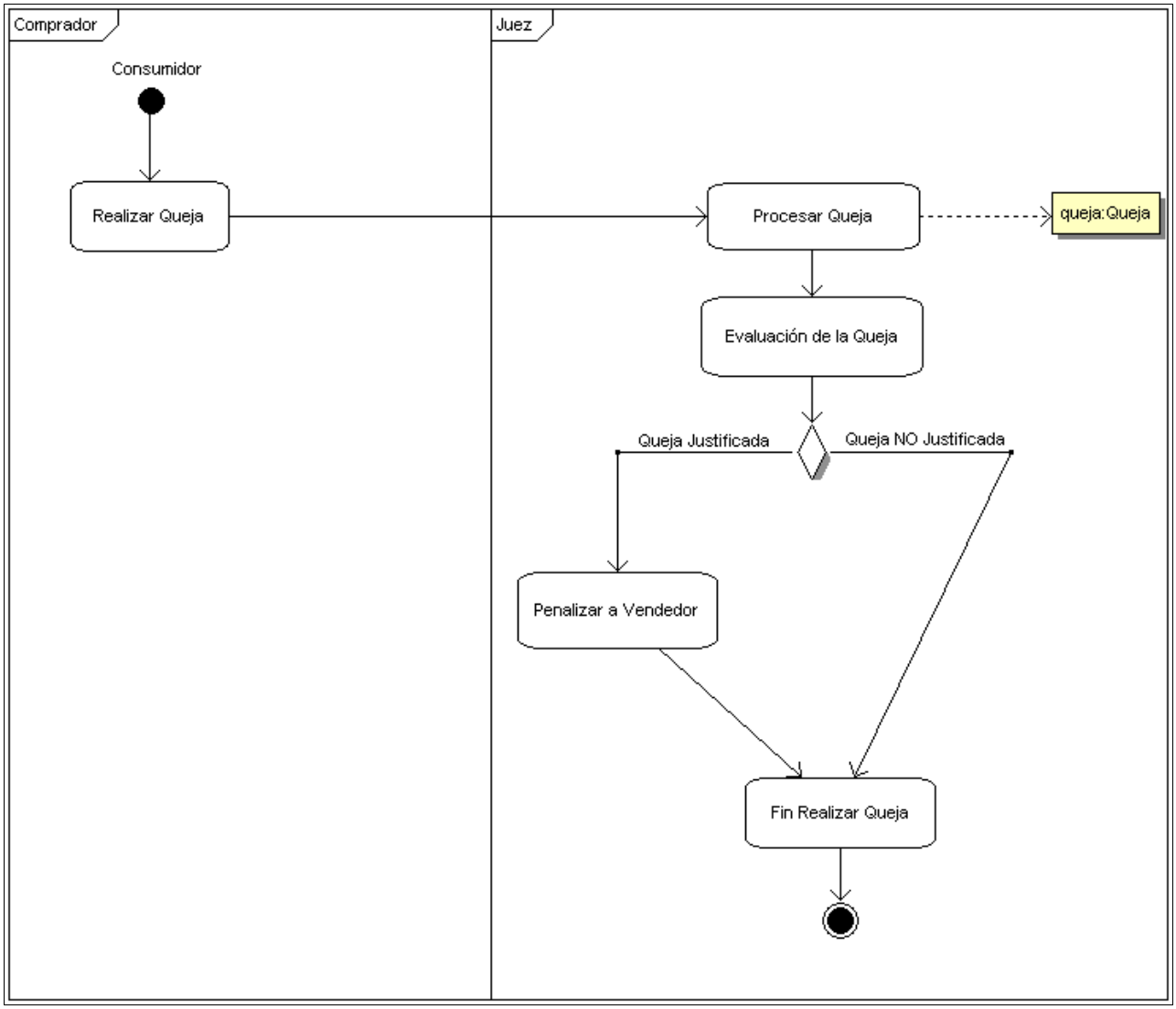


Figura 3.2.5.7: Diagrama de Actividades Realizar Queja.

Se representan las acciones que realizan un comprador (cliente que compra en MercaMadrid) y el juez del mercado para la creación de una queja sobre un vendedor del mercado.

El consumidor realiza una queja al juez por una transición realizada con un vendedor (Se crea un objeto de la clase Queja), el juez recibe la queja y la evalúa sancionando o no al vendedor acusado si la queja era justificada o no.

- ***DIAGRAMA PENALIZAR***

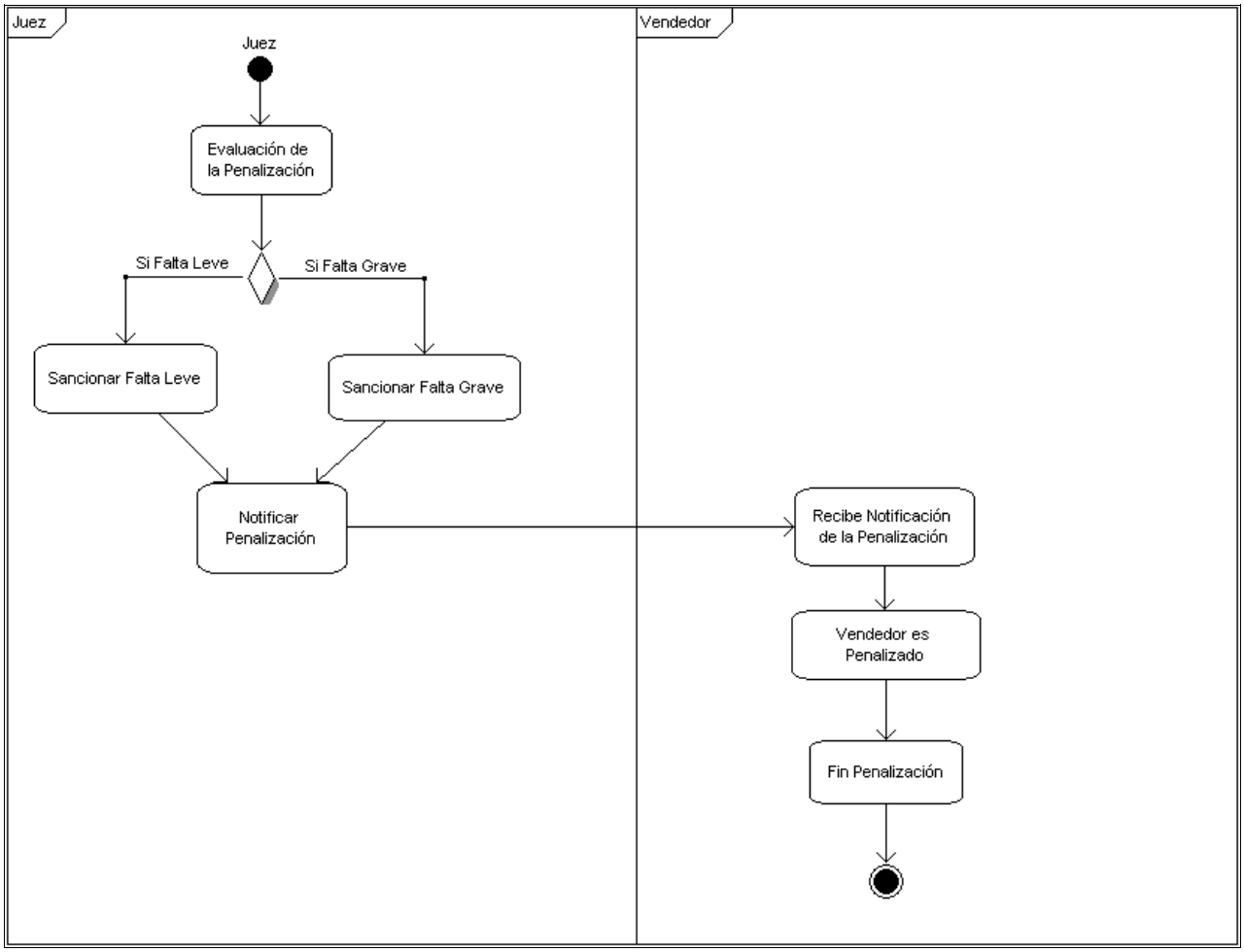


Figura 3.2.5.8: Diagrama de Actividades Penalizar.

Se representan las acciones que realizan el Juez y el vendedor (Vendedor de MercaMadrid) para la penalización del juez a dicho vendedor.

El juez evalúa la infracción cometida, decide si la falta es leve o es grave, informa al vendedor de la penalización impuesta, y lo sanciona a raíz de ella.

- ***DIAGRAMA INICIAR_APLICACION***

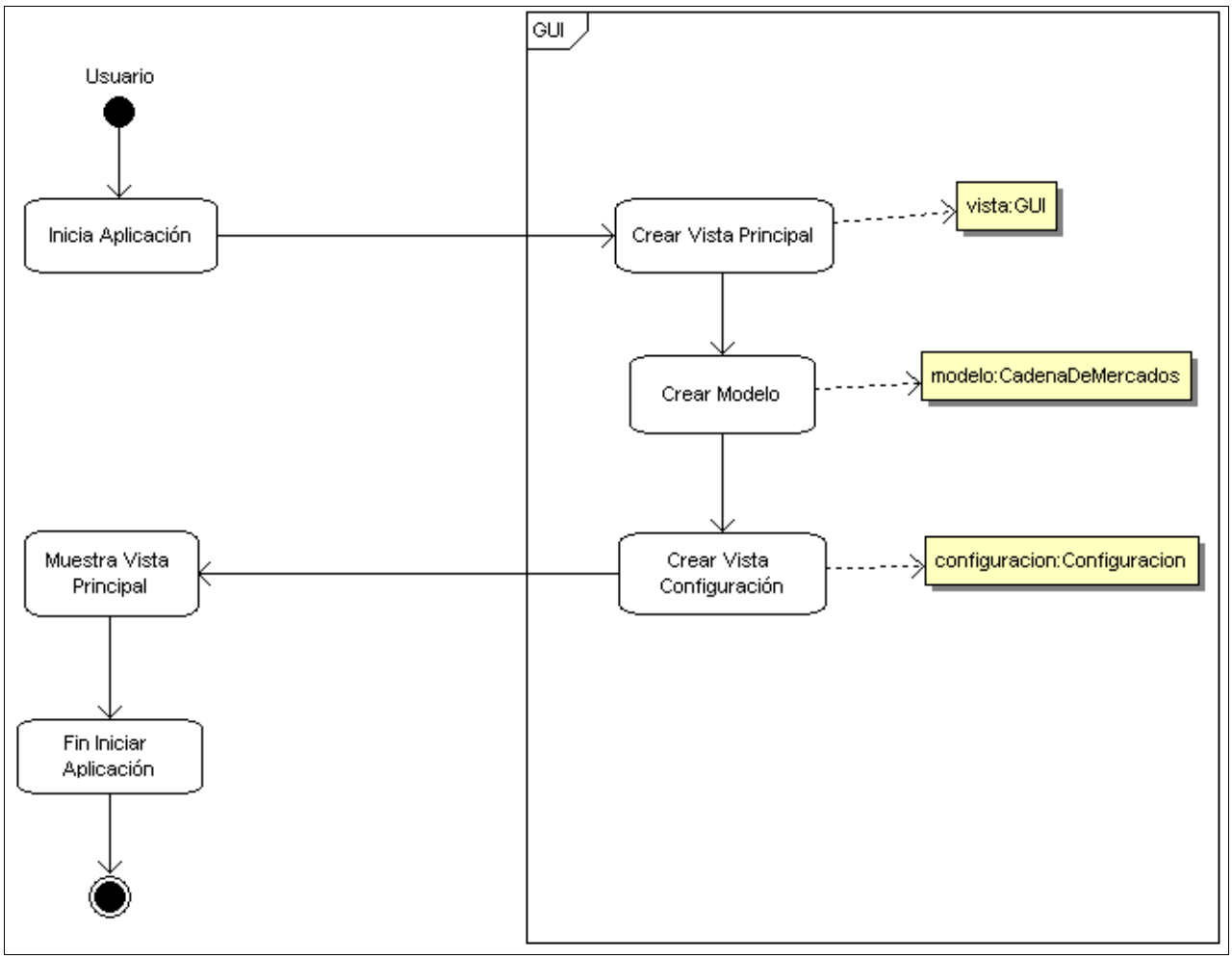


Figura 3.2.5.9: Diagrama de Actividades Iniciar_Aplicacion.

Se representan las acciones que realizan el Usuario y la GUI (vista principal de la aplicación) para la inicialización de la aplicación.

El usuario lanza la aplicación, se crean las vistas de la aplicación, se crea el modelo y se muestra la vista principal al usuario.

- ***DIAGRAMA INICIAR_SIMULACION***

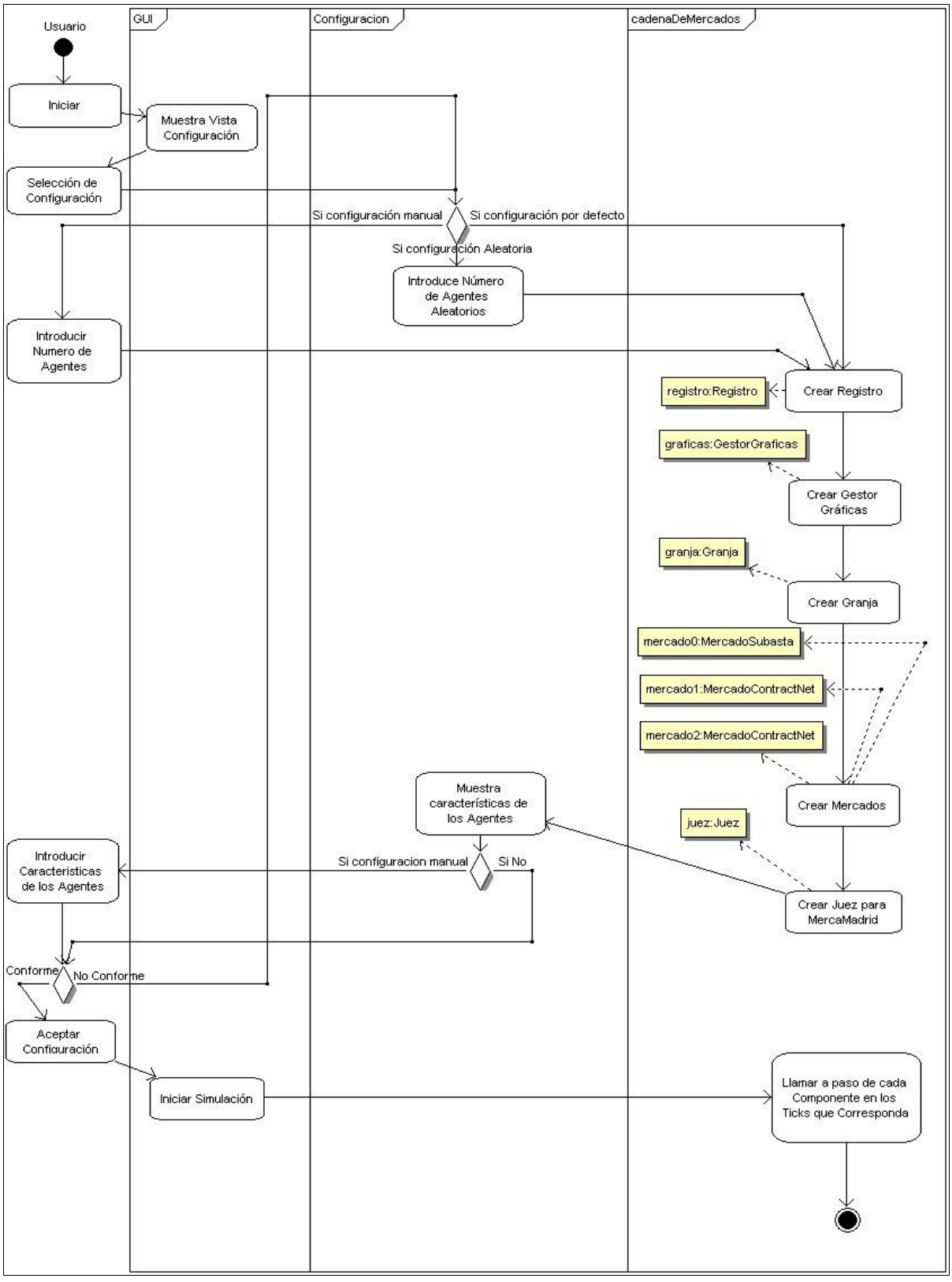


Figura 3.2.5.10: Diagrama de Actividades Iniciar_Simulacion.

Se representan las acciones que realizan el Usuario, la GUI y la Configuración (vista principal y vista de configuración de la aplicación) y la CadenaDeMercados (modelo) para comenzar la simulación de la aplicación.

El usuario inicia la simulación y se le muestra la vista de configuración. El usuario elige la configuración que desee y se crea el modelo (registro, gestor de graficas, granja, mercados y juez). Una vez el usuario acepta la configuración se inicia la simulación.

3.2.6.- Diagramas de Componentes:

En estos diagramas se reflejan las interfaces que requiere cada clase y las interfaces que la clase necesita implementar, así como las relaciones de herencia entre las clases. Además se muestra la relación física de los ficheros (debido a que el lenguaje de programación utilizado es Java la relación entre los ficheros .java y los ficheros .class).

Los diagramas son los siguientes:

- ***PAQUETE PRODUCTO***

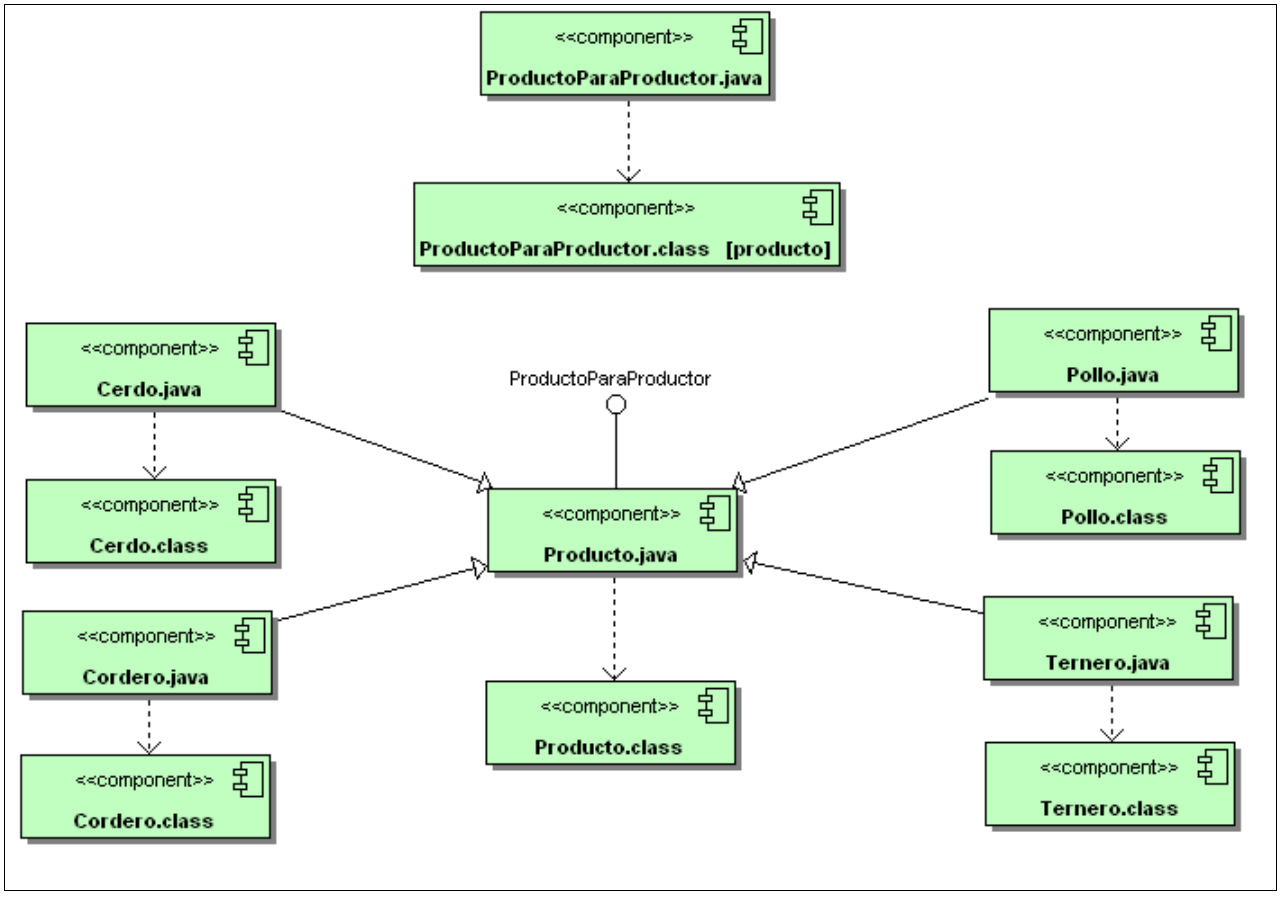


Figura 3.2.6.1: Diagrama de Componentes del Paquete producto.

- ***PAQUETE CENTROS_PRODUCCION***

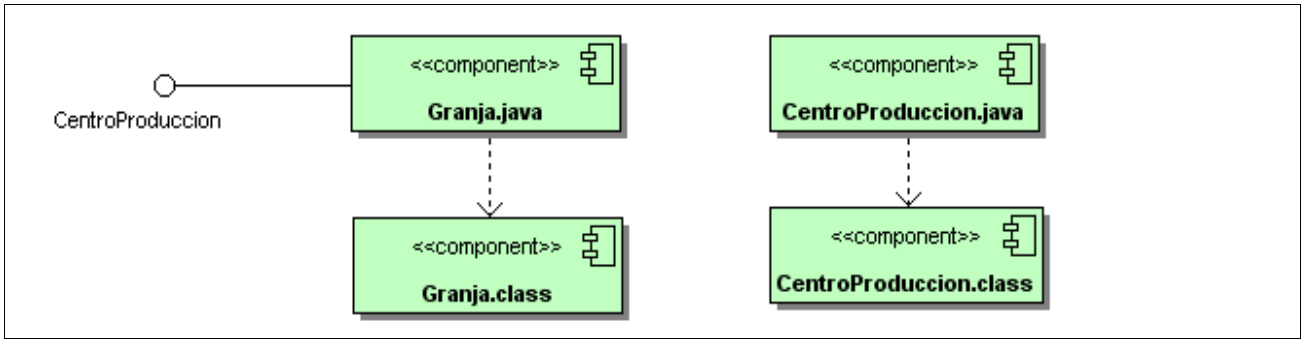


Figura 3.2.6.2: Diagrama de Componentes del Paquete centros_produccion.

- ***PAQUETE REGISTRO***

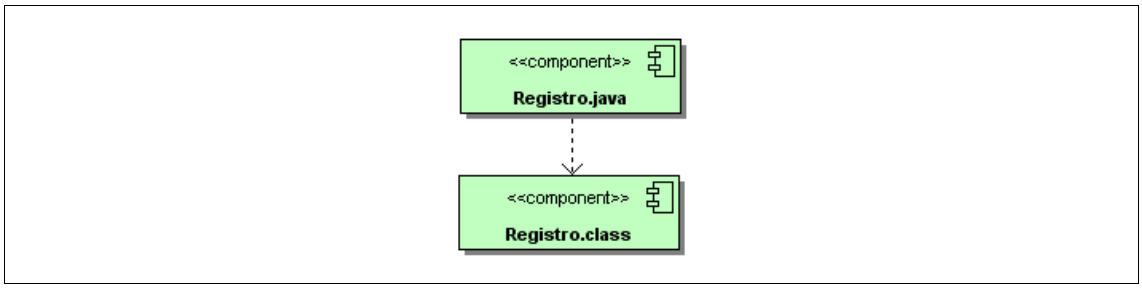


Figura 3.2.6.3: Diagrama de Componentes del Paquete registro.

- ***PAQUETE GUI***

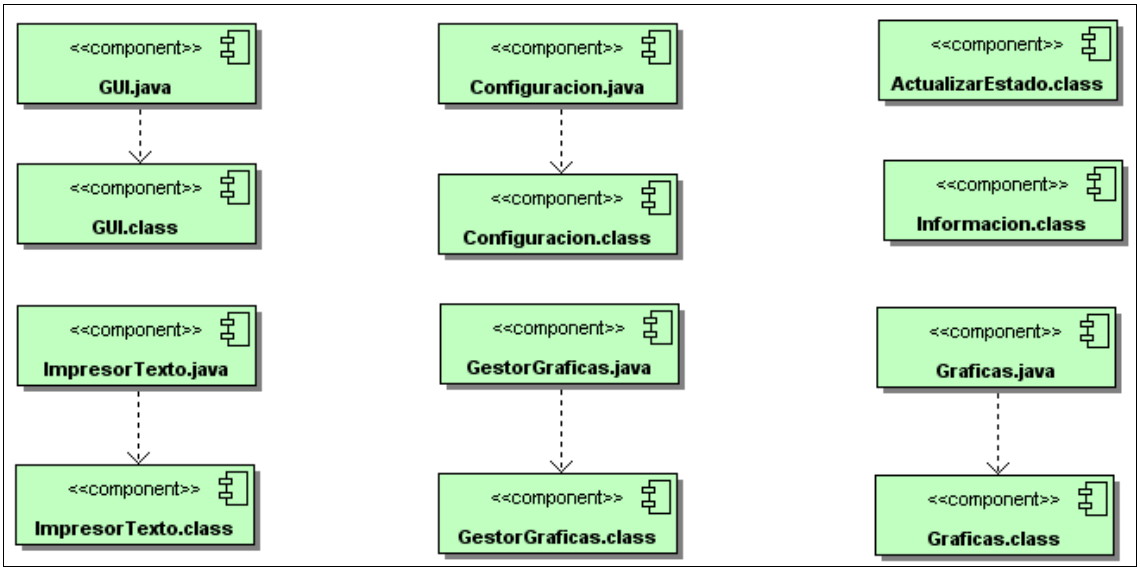


Figura 3.2.6.4: Diagrama de Componentes del Paquete GUI.

- ***PAQUETE TRANSFER***

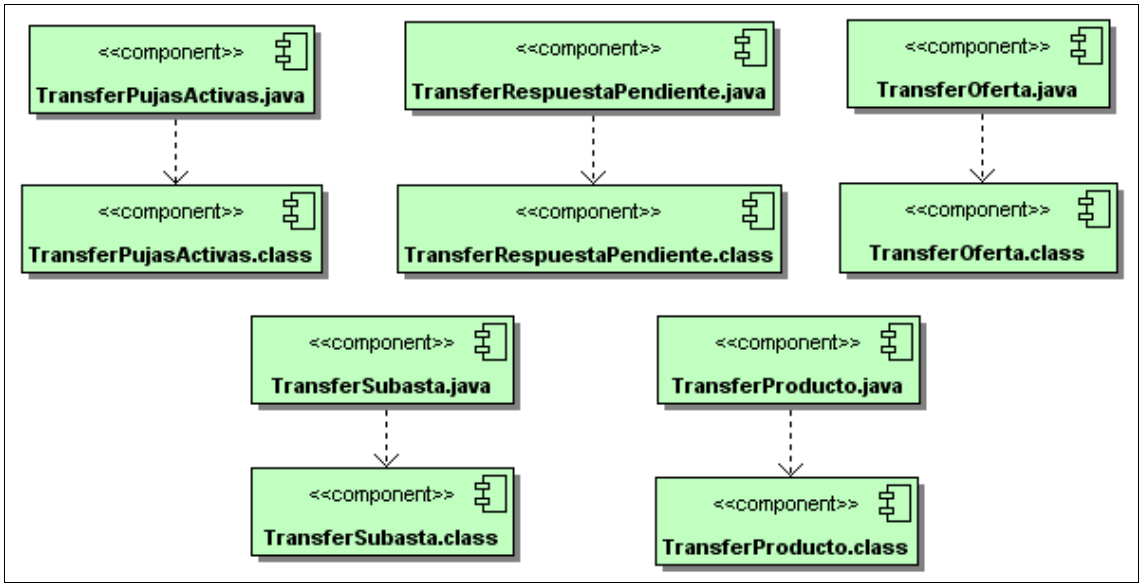


Figura 3.2.6.5: Diagrama de Componentes del Paquete transfer.

- ***PAQUETE CADENADEMERCADOS***

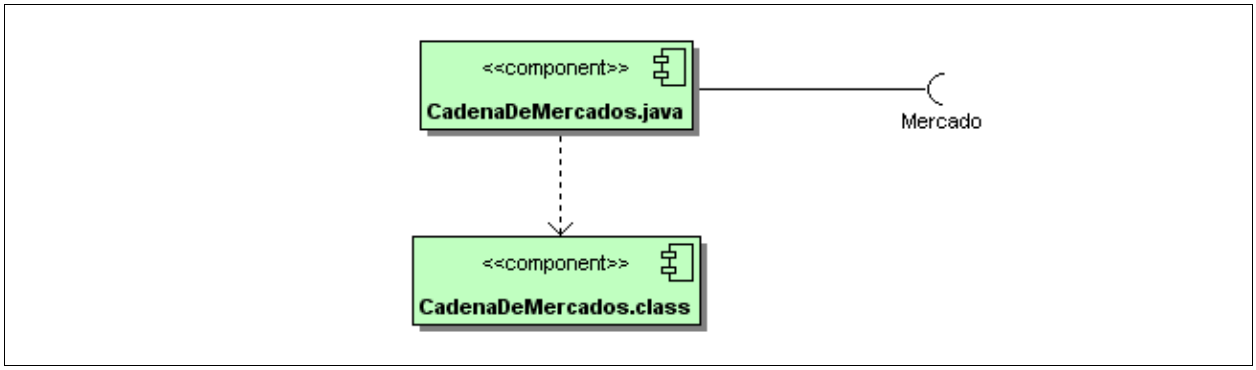


Figura 3.2.6.6: Diagrama de Componentes del Paquete cadenademercados.

- ***PAQUETE PROTOCOLOS***

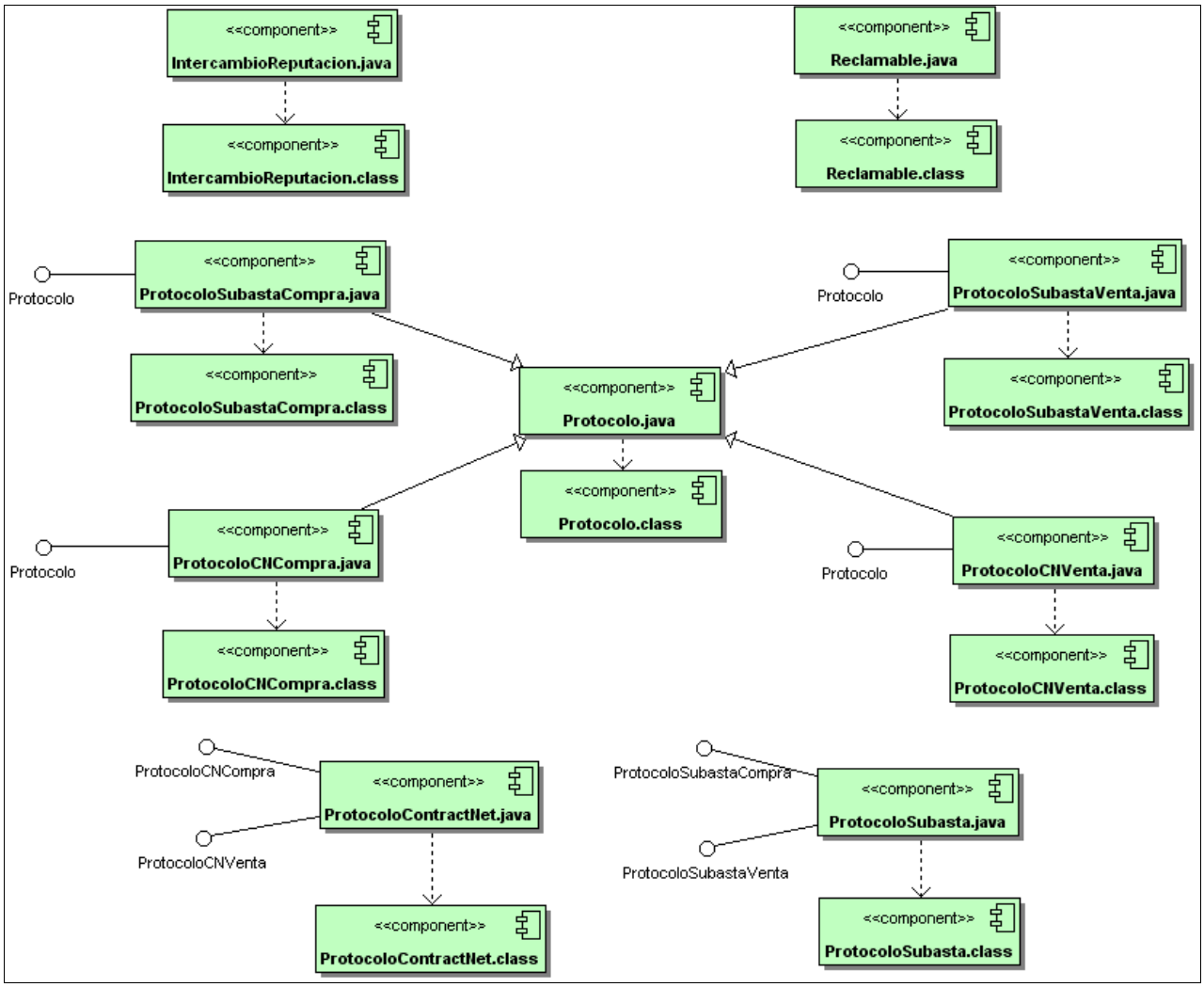


Figura 3.2.6.7: Diagrama de Componentes del Paquete protocolos.

- ***PAQUETE JUEZ***

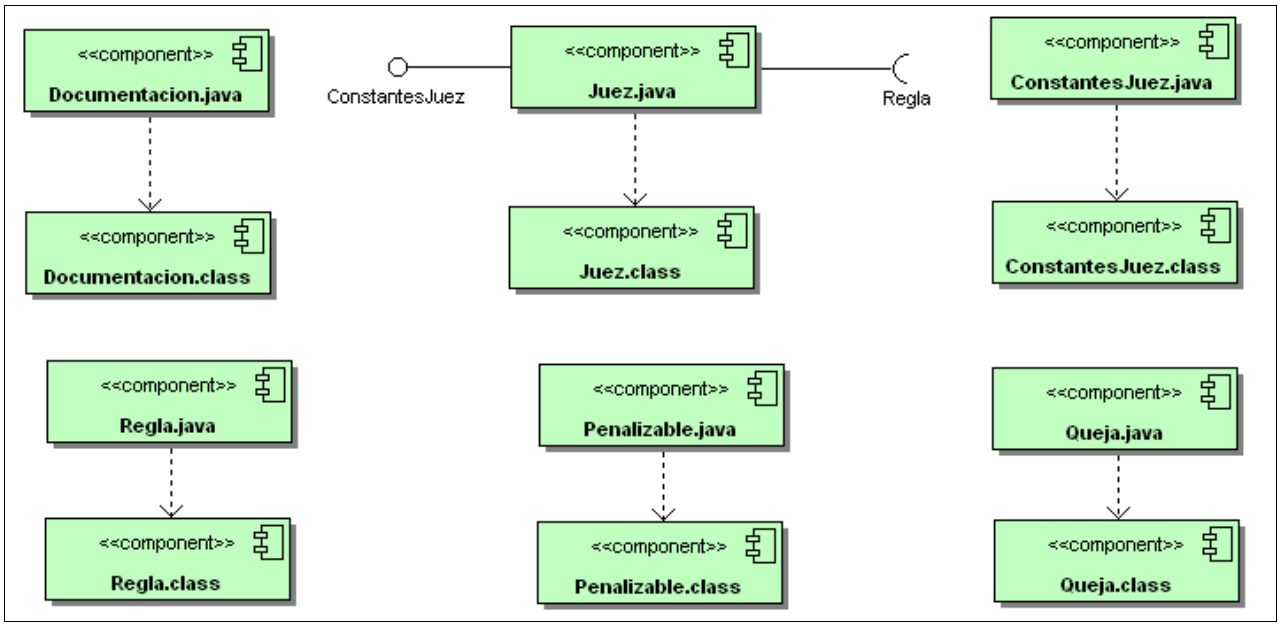


Figura 3.2.6.8: Diagrama de Componentes del Paquete juez.

- ***PAQUETE MERCADO***

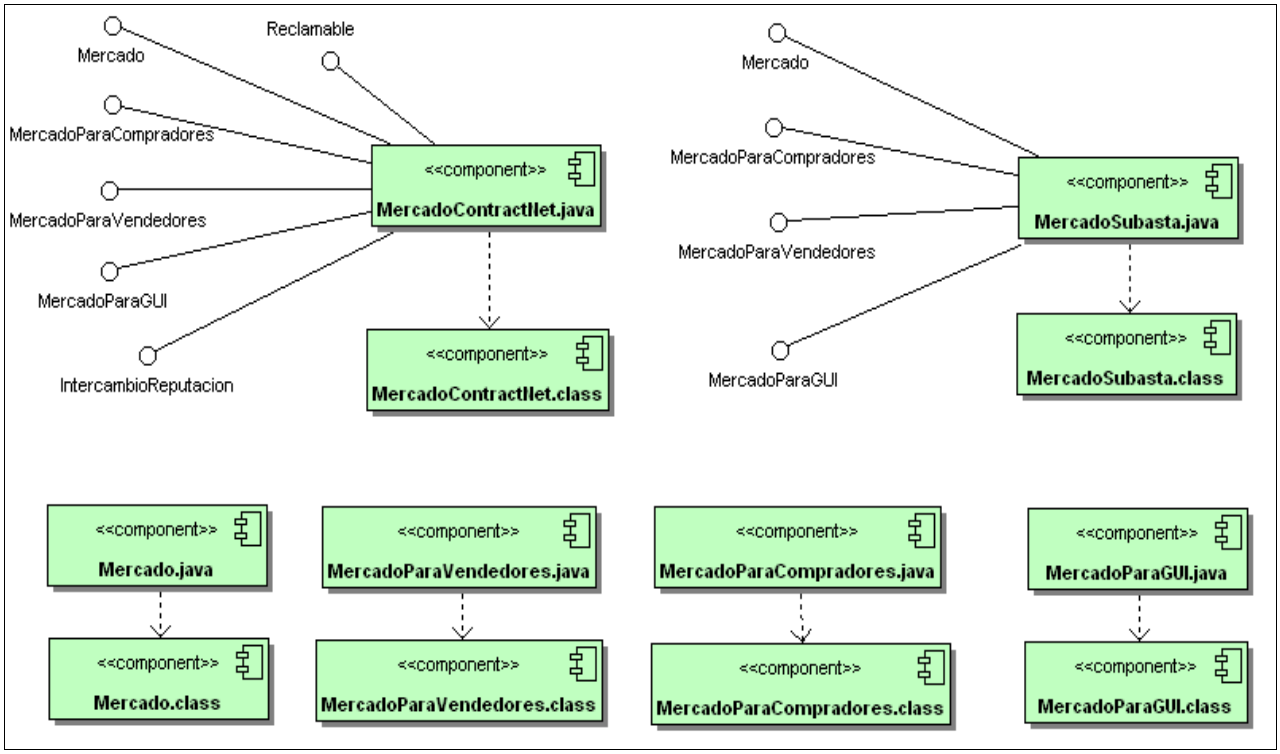


Figura 3.2.6.9: Diagrama de Componentes del Paquete mercado.

- ***PAQUETE COMPRADOR***

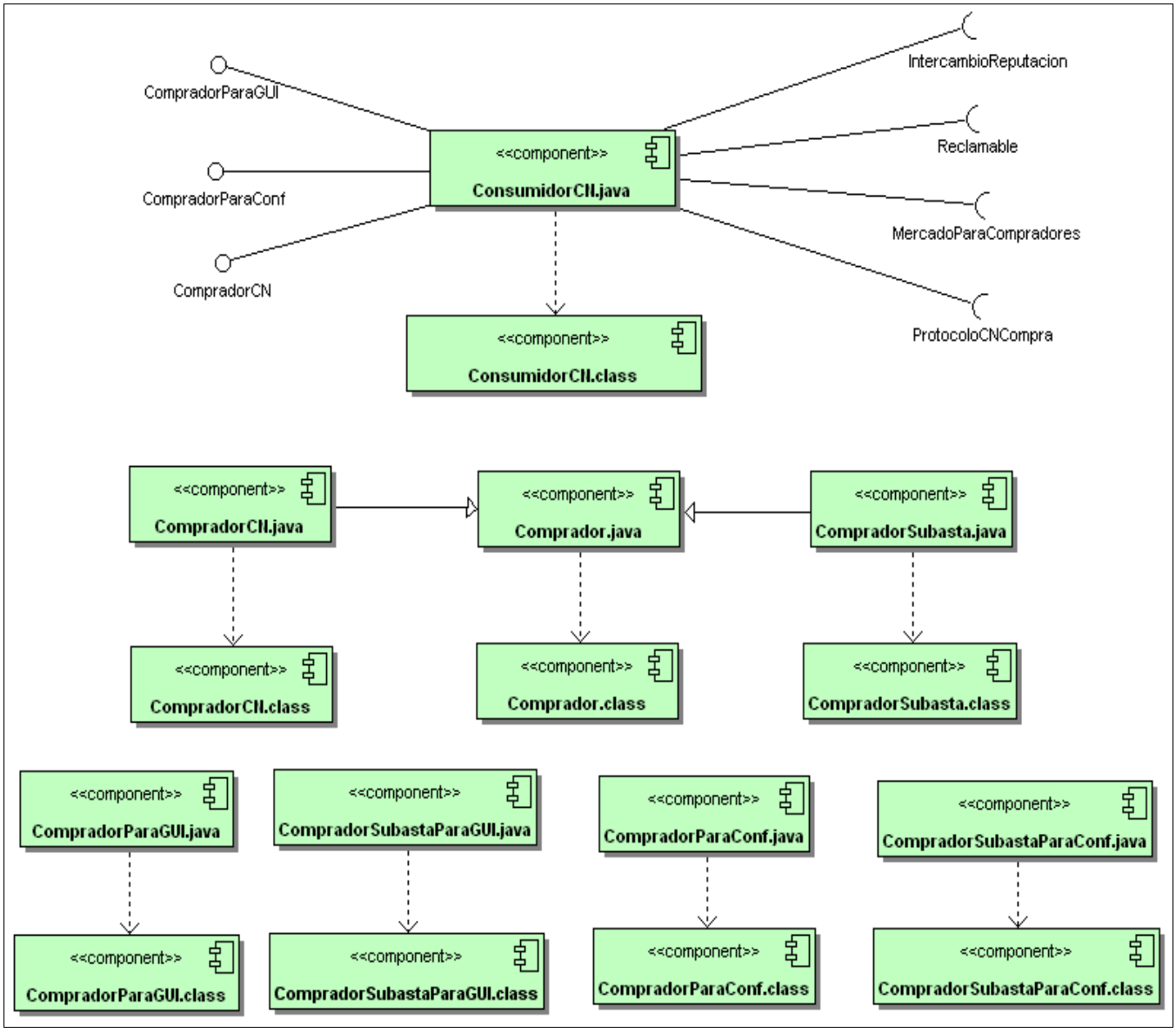


Figura 3.2.6.10: Diagrama de Componentes del Paquete comprador.

- ***PAQUETE VENDEDOR***

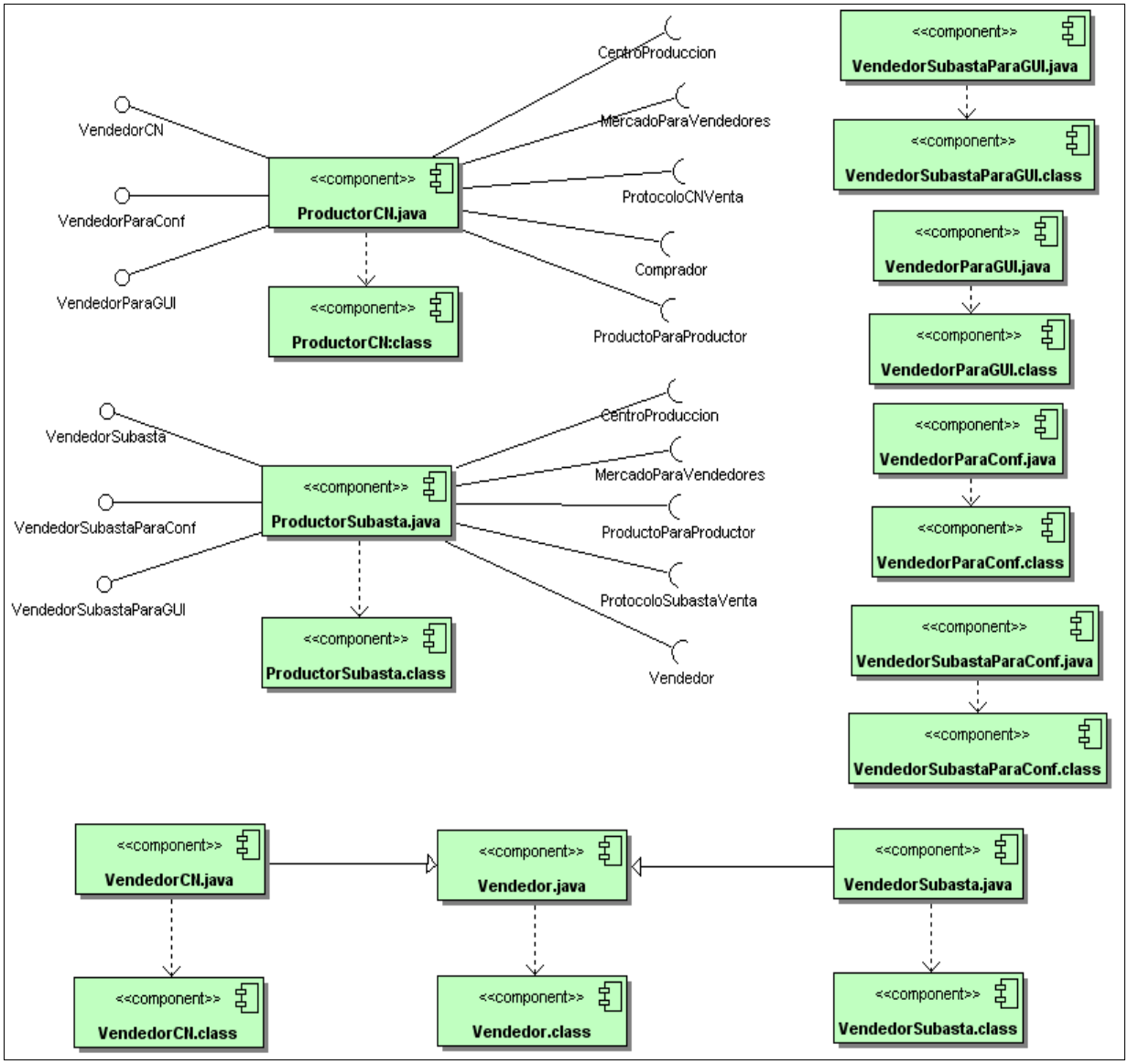


Figura 3.2.6.11: Diagrama de Componentes del Paquete vendedor.

- ***PAQUETE VENDEDORCOMPRADOR***

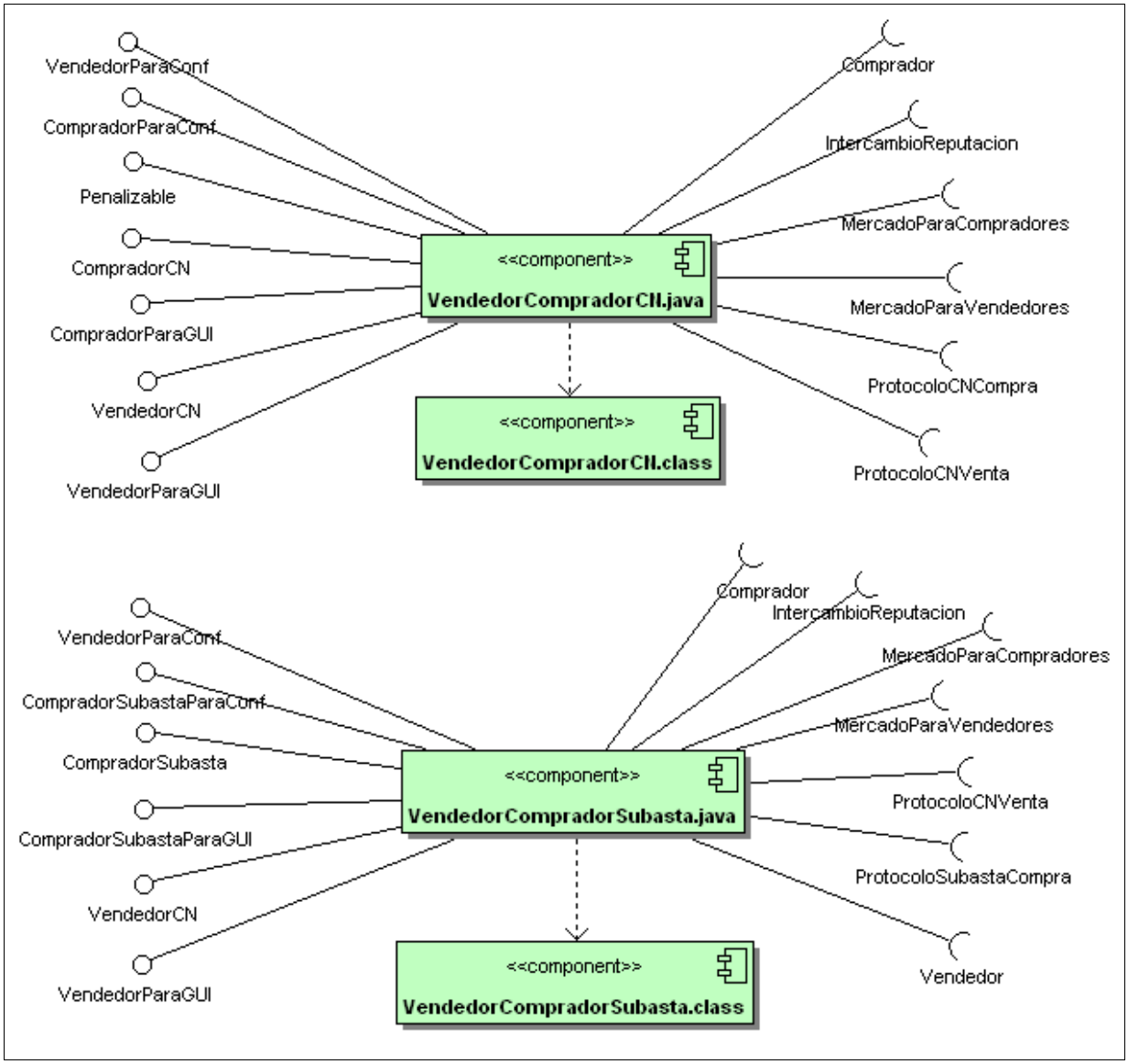


Figura 3.2.6.12: Diagrama de Componentes del Paquete vendedorcomprador.

4 *Experimentación.*

Los objetivos del proyecto no involucran llevar a cabo simulaciones para obtener resultados y conclusiones que son más del ámbito de los sociólogos o de las personas que hayan dedicado más tiempo al estudio de la reputación.

Hemos dedicado este apartado para realizar pruebas a nuestro sistema comparando los resultados obtenidos en algunas simulaciones con los obtenidos por otras personas en estudios más teóricos.

Resulta muy difícil encontrar resultados de simulaciones que sean lo suficientemente similares a las nuestras como para que sean comparables. Dentro de los numerosos estudios y artículos que hemos consultado y que aparecen listados en el apartado de bibliografía, la mayoría transcurren en el plano teórico y no ofrecen resultados de simulaciones. Dentro de estos, algunos ofrecen gráficas de los resultados esperados y descripciones y resultados de algún experimento, si bien estos datos no son muy aplicables a nuestro sistema. Otros estudios se limitan al estudio teórico y no ofrecen ningún dato experimental.

Los datos que se pueden obtener de otras fuentes que no sean estudios sobre confianza y reputación no son muy útiles para nuestro propósito. Consultas sobre mercados en el Instituto Nacional de Estadística [16]o el apartado de estadísticas de la página web de MercaMadrid[17] no ofrecen mejores resultados, puesto que los datos que proporcionan se refieren más bien a volumen de ventas y demás que no son relevantes para indicar quién y por qué ha vendido más.

Dentro de los estudios que sí que presentan resultados de simulaciones que se pueden utilizar para comparar a los de este sistema de encuentra “On the Simulation of Global Reputation Systems”[18] de Andreas Schlosser, Marco Voss y Lars Brückner. En él se presentan varias formas de computar la reputación y los resultados de aplicar esos cómputos en un sistema de un solo nivel de transacciones.

Hemos implementado algunas de las formas de tratar la reputación descritas en ese trabajo y hemos realizado simulaciones para comparar los resultados. Las diferencias de nuestro sistema con el propuesto en [18] son ostensibles, siendo la principal que en nuestro sistema, los compradores y los vendedores de MercaMadrid (como en la realidad) son agentes diferentes. No tiene sentido que un vendedor de MercaMadrid compre a su vez en MercaMadrid. Esto provoca que en el sistema de Schlosser, Voss y Brückner los agentes que venden valoren también a sus competidores cuando compran algo de

ellos y esto no sucede en MercaMadrid ni en nuestro sistema, donde un vendedor solo es valorado por los compradores y es la confianza de los compradores la que debe ganarse.

El que un agente compre de sus competidores constituye un "doble juego" que permite que en [18] se describan hasta 6 tipos diferentes de agentes en función de su actitud hacia el resto a la hora de vender y a la hora de valorarlos.

Dado que en MercaMadrid y en nuestra simulación sólo los compradores valoran a los vendedores, sólo hemos podido diferenciar dos tipos de agentes, los que cuidan que los tiempos y calidades acordados sean los que reciban los compradores (si no mejores), que llamaremos *Buenos*, y los que no cumplen las condiciones de sus tratos, que llamaremos *Malos*.

Descripción del experimento.

El experimento se desarrolla en la cadena descrita en esta memoria, que consta de 3 mercados y agentes distribuidos de la siguiente manera.

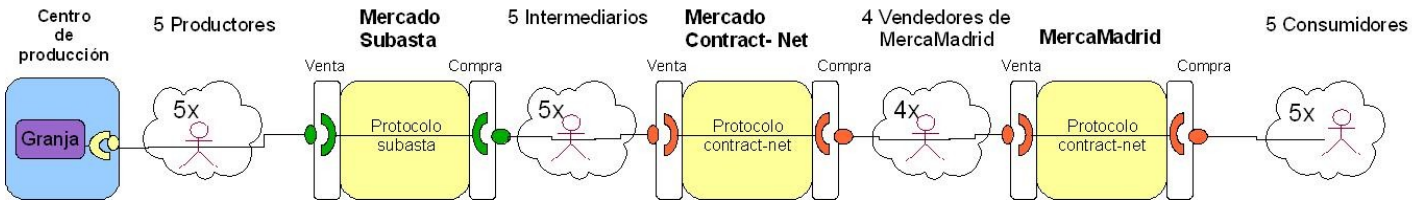


Figura 4.1 – Cadena del experimento.

- Los productores son agentes que compran productos al centro de producción y los venden al mejor postor en un mercado que opera mediante subasta. Los valores de configuración de estos agentes, de los cuales habrá cinco, serán aleatorios.
- Los intermediarios que venden estos productos a los vendedores de MercaMadrid son agentes que pujarán por los productos hasta una cantidad máxima que depende de cada uno y luego los venderán a un precio, generalmente superior, obteniendo un margen de beneficio que

también depende de cada uno. Habrá cinco intermediarios y sus valores de configuración también serán aleatorios.

- Los vendedores de MercaMadrid comprarán los productos de un intermediario que elegirán en cada momento buscando un equilibrio entre la confianza que les inspira y el precio que les ofrece. Habrá cuatro vendedores en MercaMadrid y tendrán todos unos valores de configuración iguales[1] con la salvedad de que dos serán *Buenos* y dos *Malos*.

    - Se dirá que una oferta a del vendedor x es mejor que otra b del vendedor y si:

      precio(a)*(1-confianza(x))<precio(b)*(1-confianza(y))

    - Los compradores, de los cuales habrá 5, de MercaMadrid elegirán con el mismo criterio al vendedor al que le comprarán el producto y, en el momento que lo reciban, valorarán al vendedor y transmitirán sus confianzas al comprador que lo reemplace.

    - El juez de MercaMadrid estará desactivado para asemejar las condiciones del experimento a las del experimento descrito en [18].

    - Para evitar problemas de abastecimiento, se utilizará tan solo un tipo de producto.

- Hay que destacar que ante la falta de un método estadístico para unir varias gráficas, hemos intentado elegir la gráfica que más se acercaba al resultado promedio de la simulación.

Antes de mostrar los resultados del experimento comparativo referente a la evolución de la reputación con diferentes formas de cómputo, vamos a describir un experimento realizado en las mismas condiciones que éste (las descritas arriba) y que da una idea de la variabilidad de resultados que produce una cadena compleja.

Puesto que los compradores eligen siempre al vendedor que más barato les ofrece los productos (siempre que las confianzas en ambos vendedores sean iguales), cabría esperar que en una cadena donde todos los vendedores fuesen

1 Estos valores serán los que se asignan por defecto con dos excepciones: paciencia=100 y margenBeneficio=40.

iguales, vendiesen más productos aquellos que menos margen de beneficio sacasen, puesto que serían los que ofrecerían unos precios más baratos.

Para la simulación se han utilizado los datos descritos más arriba con tres diferencias:

- Todos los vendedores son *Buenos*.
- Los margenes de beneficio ya no son todos iguales, son 20, 40, 60 y 80.
- Existirán 10 compradores en vez de 5.

A continuación se expone una gráfica que marca el valor acumulado de las ventas que ha realizado un vendedor a lo largo del tiempo. Los vendedores están etiquetados con el margen de beneficio que obtienen de cada compra.

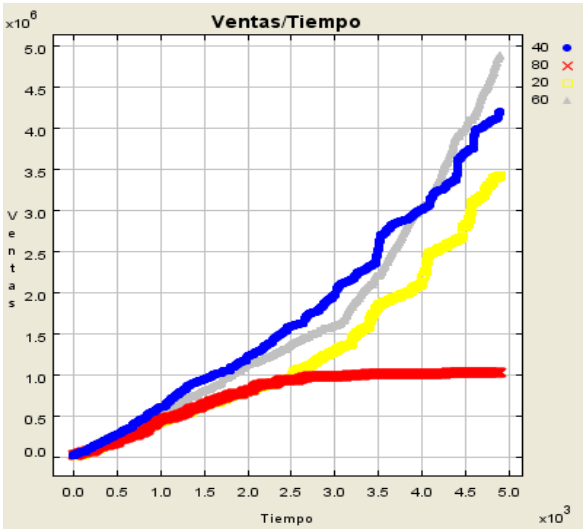


Figura 4.2 – Ventas/Tiempo.

Efectivamente, observamos que el vendedor que menos ha vendido es el que pedía un precio más alto por los productos, sin embargo, entre los tres otros vendedores el orden no es el esperado, han vendido más los que más caro vendían.

¿Por qué sucede ésto? El análisis más detallado de lo que va sucediendo en el mercado en cada momento (esto se puede hacer con la interfaz gráfica de la que dispone el simulador), nos permite encontrar la respuesta a esa pregunta.

- Cuando un comprador solicita un producto a los vendedores de MercaMadrid aquéllos que disponen de él contestan ofreciendo un precio. De entre los precios recibidos, teniendo en cuenta la confianza, el comprador elige el que más le satisface y lo compra. El problema es que, al haber muchos más compradores que vendedores, los vendedores no tienen tiempo de comprar productos al mismo ritmo que los compradores se los solicitan, dando lugar a una situación de desabastecimiento.

- De esta manera, cuando un comprador solicita un producto, lo más habitual es que a lo sumo un vendedor disponga de él, y al recibir una sola respuesta el comprador no tiene donde elegir y paga lo que le pidan, siempre y cuando no supere el límite que están dispuestos a pagar (eso es lo que le sucede al vendedor rojo, pide demasiado para algunos compradores y por tanto no consigue vender rápidamente y da tiempo al resto a que se abastezcan).

- La razón por la que los que venden más caro terminan vendiendo más es porque, sin competencia que les obligue a vender más barato, ganan más dinero, lo que les permite comprar más productos y, por tanto, vender más también.

- Esto se evitaría si los vendedores empezasen de una situación de abastecimiento o si no hubiese tantos compradores, lo que permitiría que se abasteciesen mejor. La siguiente gráfica, realizada en las mismas condiciones pero con 2 compradores solamente, lo demuestra.

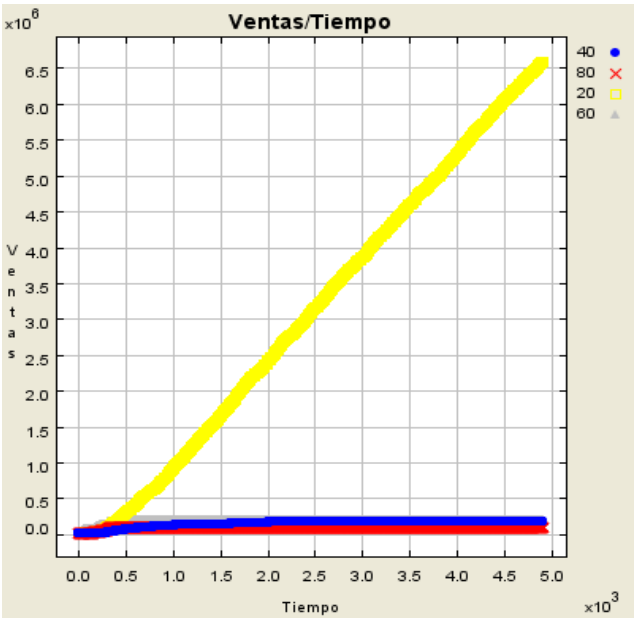


Figura 4.3 – Ventas/Tiempo.

- En la gráfica se aprecia claramente que en una situación de buen abastecimiento el vendedor que más vende es el que más baratos pone los productos. Incluso se puede ver que los demás también aparecen ordenados por margen de beneficio ascendente en la gráfica.

A continuación detallaremos los métodos de cómputo de reputación que hemos implementado en nuestro sistema y los resultados de realizar simulaciones que compararemos con los resultados obtenidos por Schlosser, Voss y Brückner.

- Sistemas acumulativos.
  - Hemos prestado especial atención a estos sistemas porque son los que mejor se adaptan a nuestro sistema dada la variabilidad de éste. Al ser una cadena compleja de la que se depende para cumplir un plazo, no siempre la buena intención de un agente va a dar un buen resultado, de manera que un sistema de valoración en el que una valoración aislada no influya mucho en el resultado es lo más apropiado.
  - Sistema EBAY.
    - El sistema EBAY consiste en que los compradores valoran con +1, 0 o -1 a los vendedores después de cada transacción. La reputación de cada vendedor es la suma de todas las valoraciones que haya recibido.

      $$rep(i) = \Sigma valoracion(i,j)$$

    - Para adaptarlo a nuestro sistema, en el que las reputaciones van de 0 a 1, hemos dividido el valor de la reputación por el número de valoraciones recibidas y hemos centrado el valor en 0.5 de manera que un reputación resultante de todas las valoraciones negativas sea 0 y la resultante de todas las valoraciones positivas sea 1.

      $$rep(i) = ((\Sigma valoracion(i,j))/\#valoraciones)*0.5 + 0.5$$

- Sistema VALUE.
  - El sistema Value se basa en el mismo método acumulativo que el sistema EBAY con la salvedad de que esta vez la valoración de cada comprador está ponderada con el valor de la transacción a la que se refiere.

$$rep(i) = \Sigma(valoracion(i,j)*coste(tj))$$

    - Para acomodarla a nuestro rango de reputaciones de 0 a 1 debemos dividir por el coste total de las transacciones del vendedor i y centrar en el 0.5.

$$rep(i) = 0.5 + 0.5*((\Sigma(valoracion(i,j)*coste(tj)))/ \Sigma coste(xj))$$

    - Los resultados obtenidos utilizando dos agentes *Buenos* y dos *Malos* con el resto de características idénticas son los siguientes:

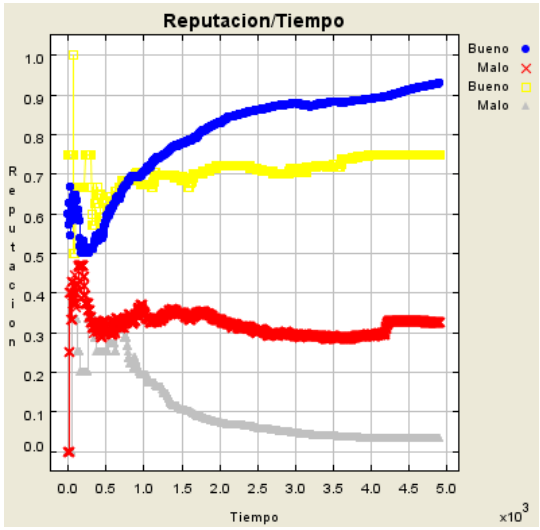


Figura 4.4 – Sistema EBAY

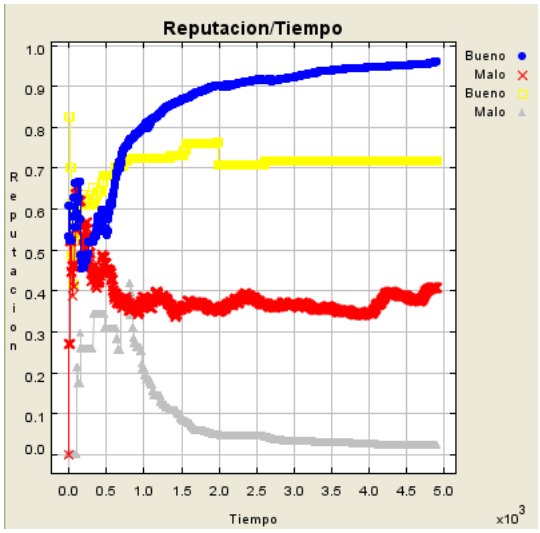


Figura 4.5 – Sistema VALUE

- A continuación se muestra el resultado presentado en [18] para una simulación con sus distintos tipos de agentes utilizando el sistema Value.

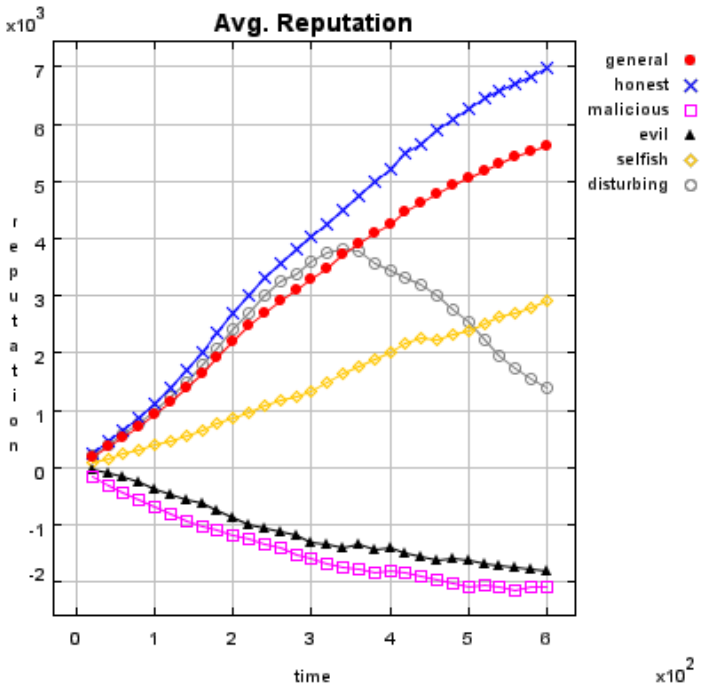


Figura 4.6 - Resultado de utilizar el sistema VALUE en [18].

- Comparando ambos resultados, se puede decir que se obtienen efectos similares en la reputación de agentes con características parecidas.

- Cabe destacar que no existe una correspondencia entre los tipos de agentes de una simulación y la otra, si bien si existen semejanzas entre algunos tipos que les llevan a obtener resultados similares.

- Se puede apreciar en las gráficas de la simulación de nuestro sistema que entre dos agentes del mismo tipo siempre el que está primero recibe mejores valoraciones. Esto se debe a la forma en la que actúan los consumidores, que ante dos ofertas iguales eligen la que primero aparece listada, que resultará ser la del agente con índice menor, provocando que cada vez que hay un conflicto el consumidor se decante por el primero, obligando al segundo a retener sus productos más tiempo, lo que reduce su calidad y a la larga su valoración.[1]

1 Cabe preguntarse si este comportamiento es realista. Sería fácil implementar consumidores que ante dos ofertas iguales eligiesen una al azar (el comportamiento de los compradores no tiene ninguna

- Sistemas de promedio
  - AVERAGE VALUE.
    - En el sistema AverageValue las valoraciones que emiten los compradores son también {-1,0,1} ponderadas por el valor de la transacción a la que se refieran, pero la reputación de un vendedor se computará haciendo la media de sus valoraciones individuales.

      $$rep(i) = ((\Sigma(valoracion(i,j))*coste(tj)))/\#valoraciones)$$

    - Este método resulta más difícil de adaptar a la escala [0,1] de reputación, por lo cual lo hemos dejado así. No obstante, es importante que si se modifica la escala de la reputación como en este caso se verifique que los agentes están preparados para manejar reputaciones en la nueva escala.

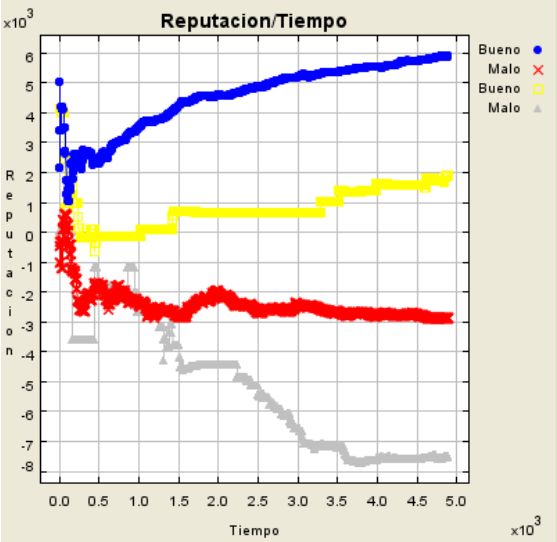


Figura 4.7 – Sistema AVERAGE VALUE.

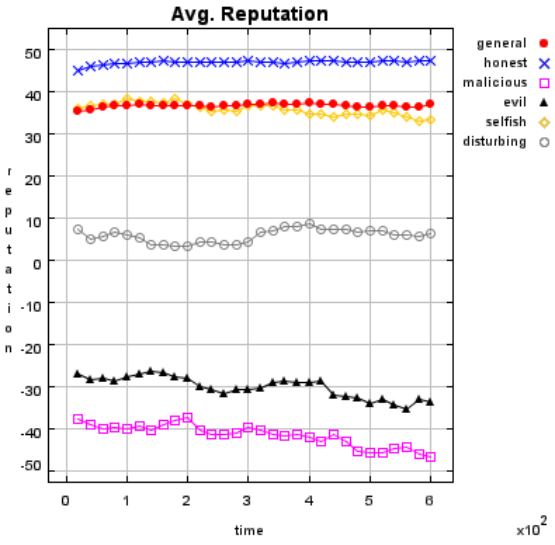


Figura 4.8 – Sistema AVERAGE VALUE en [18].

- Observando la evolución de la reputación de los agentes podemos comprobar como los agentes *Buenos* consiguen valores más altos de reputación.

limitación impuesta por el diseño), pero en la realidad tampoco es indiscutible que esto ocurra así. Hay factores tan aparentemente intrascendentes como el índice por el que se identifica al vendedor que sí influyen en las ventas de la vida real. Un claro ejemplo sería un comprador que ante ofertas iguales elija la que le ofrece el vendedor que tiene más cerca en ese momento.

- Comparando los resultados con los obtenidos por Schlosser, Voss y Brücker vemos que nuestros resultados se asemejan a partir de la mitad de la simulación a los de ellos, si bien en su gráfica no aparecen los instantes iniciales.

- En cualquier caso, este sistema se adapta muy bien a nuestro entorno, puesto que al basarse en una media, los retrasos poco frecuentes de un vendedor no repercuten en su reputación, y atenuar el efecto de esos fallos es muy importante en nuestra simulación puesto que, debido a la complejidad de la cadena, son frecuentes.

- Sistemas "borrosos" (Blurred)

  - Estos sistemas se basan en que la reputación es la suma de todas las valoraciones recibidas pero ponderadas de tal manera que tengan mayor importancia las más recientes.

  - Nuevamente las valoraciones pueden ser -1, 0 o 1 sin ponderar con el coste de la transacción.

  - Hemos utilizado una versión ligeramente distinta pero con la misma idea que [18].

  - Para computar la reputación utilizaremos la siguiente fórmula.

$$rep(i) = \Sigma(valoracion(i,j)*\lambda^{j}) \qquad \text{con } \lambda=0.95$$

    - Para adaptarla a la escala de [0,1] bastaría con dividir por el número de valoraciones recibidas y después centrar en 0.5, pero puesto que una valoración negativa puede hacer bajar mucho una reputación buena, y viceversa, la reputación es bastante variable, con lo cual no crece nunca mucho, y al dividir por el número de valoraciones tiende a hacerse 0.

    - Por tanto, hemos elegido usar este sistema sin adaptar a [0,1]. Éste es el resultado obtenido por nosotros comparado con el mostrado en [18].

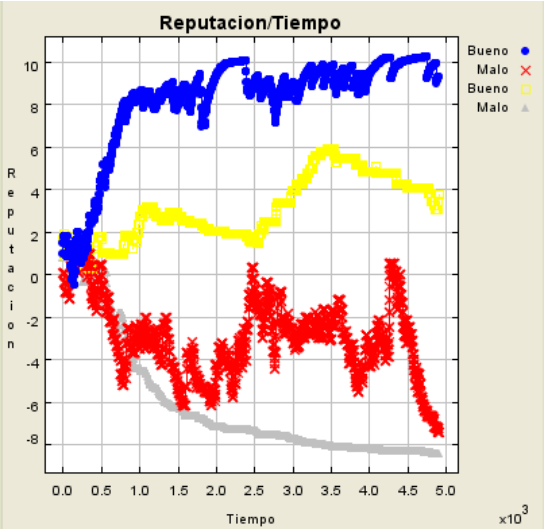


Figura 4.9 – Sistema BLURRED.

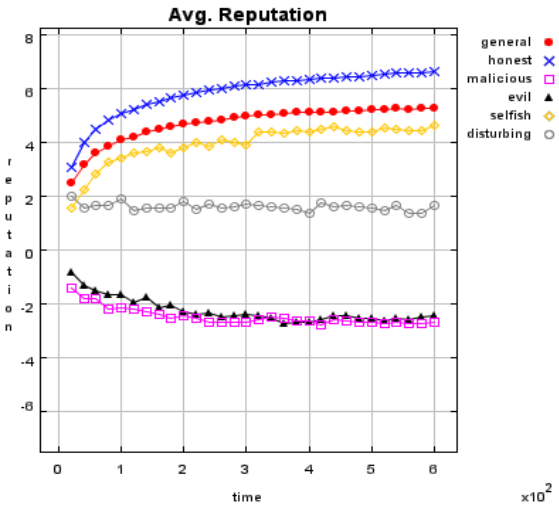


Figura 4.10 – Sistema BLURRED en [18].

- En esta ocasión, el comportamiento más variable de nuestros agentes demuestra mejor el efecto de utilizar el sistema de cómputo Blurred. Se puede apreciar que, dado que las valoraciones más recientes tienen mayor peso, las gráficas no son tan lineales (tienen pequeñas "caídas") cuando existen valoraciones que se apartan de la media.

- Este método se adapta mal a nuestro sistema, puesto que una mala transacción en un momento dado puede dejar la reputación del agente mermada (o viceversa), dando lugar a situaciones, si coinciden un "bache" de un agente *Bueno* con un "pico" de un agente *Malo*, en las que, momentáneamente, la reputación es engañosa.

- Sistemas OnlyLast.

  - Como su propio nombre indica, este método solo informa de la última valoración obtenida, no acumula ningún registro de valoraciones pasadas. Para acomodarlo a la escala [0,1] basta con centrar en el 0.5.

  - Se puede entender como un caso extremo de sistema Blurred con $\lambda=0$.

  - Este método solo da una idea del estado momentáneo del agente y es completamente ineficaz en un sistema variable como el nuestro.

- En un sistema en el que los comportamientos son constantes, sin embargo, este método puede ser una solución de bajo coste para computar la reputación.

- La caótica gráfica obtenida da una idea de lo variable que es la reputación con este sistema.

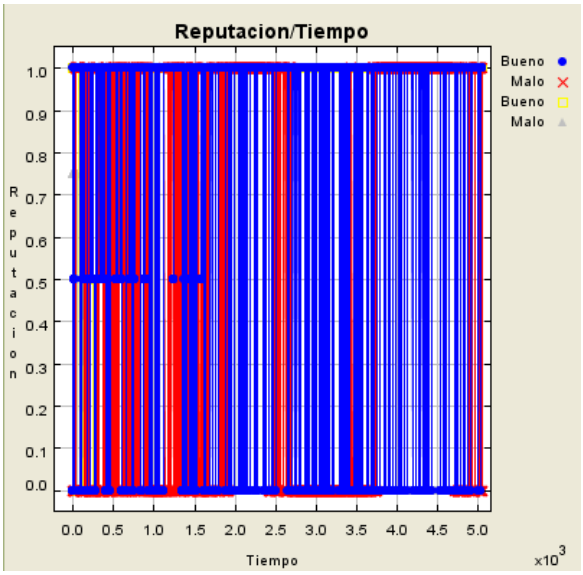


Figura 4.11 – Sistema ONLY-LAST.

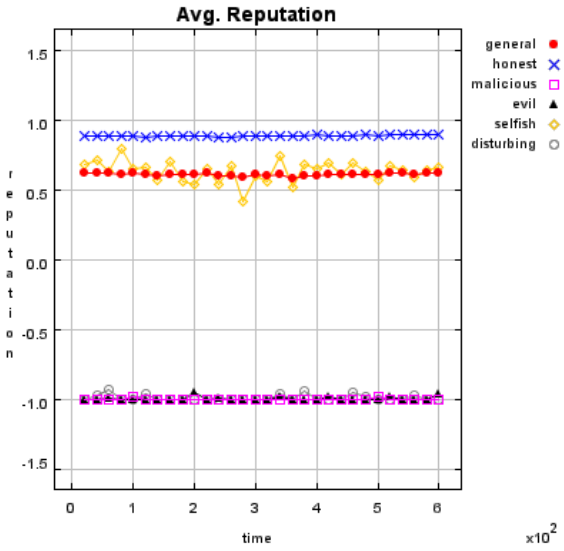


Figura 4.12 – Sistema ONLY-LAST en [18].

- La gráfica que muestran Schlosser, Voss y Brücker como resultado de la simulación OnlyLast no parece corresponderse con la versión del método que describen donde las valoraciones son solo -1, 0 o 1 y no se tiene en cuenta más que la última. Si se hubiese utilizado ese método para dibujar la gráfica no debería haber valores distintos de -1, 0 o 1. Por tanto, la gráfica se debe referir a una variación de este método.

- Sistemas adaptativos.

  - Sistema Yu-Singh.

    - Estos sistemas funcionan de manera que la variación que provoca en la reputación una valoración depende del valor de la reputación en ese momento. El sistema Yu-Singh está pensado para que sea más difícil construir una buena reputación que bajarla, puesto que cuando la reputación es alta una valoración baja la reduce significativamente.

- La reputación se calcula según estas fórmulas:

$$rep(i) = \Phi(rep(i-1), valoración(i,j))$$

donde

$$\Phi(r,q)=\begin{cases}\alpha & \text{if } r=0\wedge q>0,\\ \beta & \text{if } r=0\wedge q<0,\\ r+\alpha(1-r) & \text{if } r>0\wedge q>0,\\ \frac{r+\beta}{1-\min\{|r|,|\beta|\}} & \text{if } r>0\wedge q<0,\\ \frac{r+\alpha}{1-\min\{|r|,|\alpha|\}} & \text{if } r<0\wedge q>0,\\ r+\beta(1+r) & \text{if } r<0\wedge q<0.\end{cases}$$

donde α es la valoración positiva (en nuestro caso 0.05) y ß es la valoración negativa (en nuestro caso -0.3).

- El resultado obtenido es el siguiente:

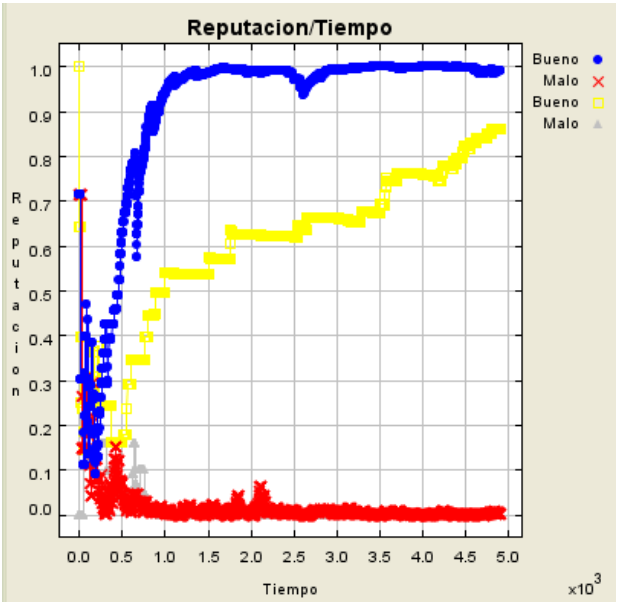


Figura 4.13 – Sistema YU-SINGH.

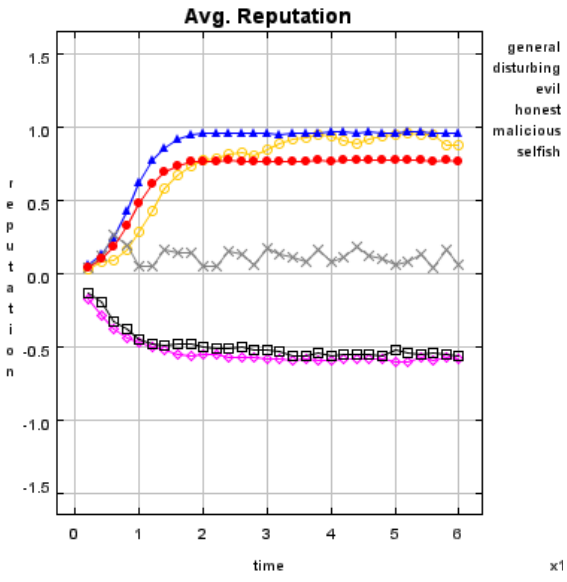


Figura 4.14 – Sistema YU-SINGH en [18].

- En la gráfica se puede apreciar como inicialmente a los vendedores *Buenos* les cuesta más alcanzar una buena reputación, pero al transcurrir un tiempo se distancian enormemente de los *Malos*. Este método es muy apropiado para nuestro sistema, porque identifica inequívocamente a los agentes que tienen un mal comportamiento reiterado diferenciándolos de los que se comportan bien

generalmente. Incluso los vendedores que actúan bien siempre que pueden pero que tienen menos oportunidades que los demás (en este caso el amarillo por la forma de elegir entre ofertas iguales explicada anteriormente) acaban obteniendo una reputación muy alta.

- Cabría plantearse si para nuestro sistema sería más apropiado un método que permitiese ganar reputación rápidamente, pero que evitase perder una buena parte por una mala transacción aislada. El utilizar el método YuSingh con α=0.3 y ß=-0.05 parece una opción adecuada. El resultado obtenido es éste:

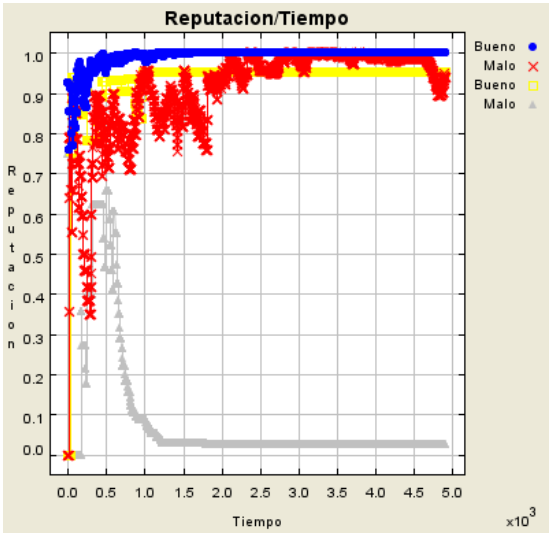


Figura 4.15 – Sistema YU-SINGH invertido.

- La gráfica muestra como un vendedor *Malo* que tiene facilidad en las transacciones puede ganarse una alta reputación a pesar de su comportamiento perjudicial para el comprador, por tanto con esos valores de α y ß el método YuSingh es muy poco apropiado para nuestro sistema.

- Esto demuestra que este para aplicar este método primero deben estudiarse los valores de α y ß que resultan más adecuados en cada caso.

El trabajo de Schlosser, Voss y Brücker solo ofrece resultados referentes al estudio de la reputación. No hemos encontrado ningún trabajo que ofrezca

gráficas relativas a este aspecto de la aplicación. Se pueden realizar experimentos con este simulador, pero ante la falta de datos para comparar, hemos decidido no incluir ninguno en esta memoria.

Respecto al uso del juez para resolver conflictos, esta es una técnica relativamente emergente e inexplorada y tampoco hemos sido capaces de encontrar resultados de trabajos previos con los que comparar el comportamiento de nuestro juez ni su influencia en la reputación de los agentes y en la confianza que unos depositan en los otros.

5 . Conclusiones

5.1 Lecciones aprendidas

A lo largo del año, hemos desarrollado varias versiones de sistemas multi-agente con distintos diseños (centralizado y distribuido). Esto nos ha permitido conocer el funcionamiento de dichos sistemas y realizar un acercamiento al campo de la simulación con agentes.

En primer lugar, hemos comprendido que la programación con agentes es una herramienta muy potente que nos permite crear un entorno computacional e introducir, en él, agentes software que colaboren entre ellos, actúen en solitario, regulen el comportamiento del sistema, etcétera... según el propósito del programa. Asimismo, hemos aprendido los tipos de agente que pueden existir y con qué finalidades se crea cada uno de ellos, dependiendo no solo de su propósito sino también del entorno en el que serán introducidos. Desde un agente autónomo que trabaja de forma no colaborativa y sin ningún tipo de aprendizaje, hasta agentes inteligentes que pueden llegar a describir el comportamiento de sujetos reales con una precisión útil y coexisten en un sistema utilizando protocolos de comunicación para colaborar y alcanzar objetivos comunes.

En lo que se refiere al campo de la simulación con agentes, hemos trabajado especialmente. Durante nuestro trabajo, hemos aprendido a modelar el comportamiento de los agentes para obtener unos resultados significativos en la simulación del sistema. Hemos aprendido a utilizar la herramienta RePast que nos ha provisto de una base sobre la cual trabajar en la simulación mediante el método step, así como para generar las gráficas que hemos utilizado para analizar en detalle el comportamiento de la simulación y generar unas conclusiones interesantes.

Por otra parte, hemos mantenido un estrecho contacto con el sistema ART (Agent Reputation Trust). Durante el extenso periodo de documentación y trabajo en el proyecto, hemos aprendido que mediante la incorporación de la confianza y la reputación en el sistema, los agentes son capaces de detectar a aquellos agentes poco fiables o fraudulentos y aislarlos del sistema. La confianza y la reputación es un acercamiento al comportamiento real de las sociedades que nos ha permitido mejorar la fiabilidad de nuestro sistema y crear un entorno en el que el fraude es penalizado.

5.2 Trabajos Futuros

Como se comento anteriormente, la aplicación describe el funcionamiento del mercado mayorista Mercamadrid, incluyendo además toda la cadena de distribución de los productos desde que son fabricados, hasta que llegan a los compradores de Mercamadrid.

La aplicación que hemos realizado está diseñada para ser ampliada en el futuro ya que se pueden implementar nuevos tipos de centros de producción de productos, productos con diferentes características a los existentes, nuevos protocolos de compraventa entre agentes, etc.

Con esta base, un programador, podría añadir nuevos elementos de los anteriormente mencionados siguiendo la estructura diseñada para la aplicación y así asemejarla más aun a lo que ocurre en la realidad y poder obtener unos resultados de simulación más completos.

También el programador podrá añadir nuevos comportamientos a los agentes de la aplicación y nuevos sucesos en la cadena de distribución para que las simulaciones sean todavía más realistas.

- Ampliaciones a nivel estructural y de diseño:

En la aplicación se manejan cuatro productos diferentes (Cerdo, Pollo, Ternero y Cordero), que tienen las mismas características. A pesar de esto, la aplicación está diseñada de tal manera, que se pueden añadir nuevos productos con diferentes características a los ya existentes sin necesidad de modificar la estructura del programa.

Los productos añadidos pueden ser arbitrariamente complejos, con características o comportamientos propios.

Para añadir nuevos productos (o familias de éstos), el programador tendrá que crear una nueva clase por cada producto nuevo que incluya atributos y métodos para modelar sus características. Estas nuevas clases para los productos tendrán que heredar de la clase Producto y que tenga el constructor que llama a la clase padre pasándole el producto que es, el identificador del productor al que pertenece el producto y el tiempo que falta para que el producto esté listo.

Además por cada producto se deberá añadir a la clase producto un identificador para ese producto.

La aplicación está construida con un solo centro de producción de productos, una granja, pero está preparada para añadir nuevos. Así se podrían crear campos de cultivos, bancos de pesca, nuevas granjas, etc.

Para ello, el programador tendrá que crear una nueva clase que sea el nuevo centro de producción que implemente la interface CentroProduccion.

Además tendrá que tener el conste de fabricación y el coste de tiempo de fabricado de cada uno de los productos que se puedan crear en el nuevo centro de producción. También el nuevo centro tendrá que tener un puntero al registro único de la aplicación para dejar constancia de sus transacciones y al ImpresorTexto para mostrar al usuario por consola dichas transacciones.

Para la simulación de las operaciones de compraventa entre los agentes, se han implementado dos protocolos diferentes, el protocolo Contract Net y el protocolo subasta del tipo inglesa. En la realidad, durante la cadena de distribución pueden aparecer nuevos tipos de protocolos de compraventa como por ejemplo en el comercio de las lonjas de pescado que se suele utilizar como forma de compra la subasta holandesa o subasta inversa, es decir, se fija un precio inicial y este va descendiendo.

Para añadir un nuevo protocolo de compraventa, el programador tendrá que crear dos interfaces una para las operaciones de compra y otra para las operaciones de venta del nuevo protocolo y que las dos hereden de la interface existente Protocolo. Luego tiene que crear una clase para el protocolo que implemente las dos interfaces que creo para este. En esta clase se implementaran los métodos necesarios para la realización de las operaciones de compraventa a través del nuevo protocolo, para lo cual, el protocolo tendrá que contener un mercado en el que los compradores y los vendedores utilicen este nuevo protocolo.

Por todo esto, al añadir un nuevo protocolo, el programador tendrá que crear una nueva clase que sea el mercado del tipo del nuevo protocolo que tendría que implementar las interfaces Mercado, MercadoParaCompradores, MercadoParaVendedores, MercadoParaGUI y MercadoParaConf; además de añadirle las operaciones necesarias para poder realizar la compraventa de productos.

Para que pudieran comprar en este nuevo mercado, el programador tendría que crear nuevos compradores y vendedores que utilicen el nuevo protocolo de venta. Así, la clase nueva del comprador tendría que implementar la interface Comprador y unas nuevas interfaces creadas por el programador para las operaciones del protocolo, para la vista y para la vista de configuración del nuevo

comprador; mientras que la nueva clase vendedor tendría que implementar la interface Vendedor y unas nuevas interfaces creadas por el programador para las operaciones del protocolo, para la vista y para la vista de configuración del nuevo vendedor.

Con todo esto, como ya se comentó en el apartado de diseño, la clase del nuevo protocolo haría de fachada entre el nuevo mercado y los nuevos compradores y vendedores que utilizan este protocolo de compraventa.

También se pueden añadir nuevos protocolos para la comunicación entre agentes para realizar coaliciones entre ellos para obtener algún beneficio (trabajar en equipo). Para ello bastaría con que el programador creara una interface que tuviera los métodos necesarios para realizar dicha comunicación y que los tipos de agente que realicen dichas acciones implementen dicho interface desarrollando dichos métodos.

Otra ampliación, ya comentada en el apartado de adaptación, consistiría en convertir la cadena de mercados en un entorno dinámico permitiendo que los agentes cambien de mercado durante la simulación. Para ello sería necesario describir un protocolo para realizar estos cambios, en los que estarían involucrados tanto agentes como mercados, y poner a disposición de los agentes una lista con los mercados disponibles.

- Ampliaciones a nivel de comportamiento:

A nivel de funcionalidad se pueden añadir nuevos comportamientos de los agentes y nuevas situaciones con las que se conseguirían simulaciones más completas.

Se podría implementar que los vendedores que tengan un producto del que no haya mucha cantidad, se alíen entre ellos para subir los precios de ese producto y encarecerlo, es decir, una especie de monopolio que repercutiría muy positivamente en su economía.

Para contrarrestar esto, al juez debería de añadírsele más funcionalidad para que pudiera detectar los posibles monopolios viendo que el precio de un determinado producto está subiendo en demasía sin causa aparente. Una vez que el juez detectara un intento de alianza, este sancionaría a los agentes vendedores que han cometido esta infracción.

También se podrían implementar coaliciones de agentes a la hora de realizar las transacciones de intercambio de reputación, es decir, cuando un

comprador le pida a otro comprador la confianza que él tiene en un vendedor, el comprador al que se la solicito podría mentir diciendo que su confianza es más alta de lo que en realidad es si entre el comprador y el vendedor han llegado a un pacto para que hable bien de él a cambio de precios más baratos.

Por el contrario, también se podría realizar que al solicitarle la confianza, el comprador no se comporte bien y diga porque sí que su confianza en el vendedor es muy baja cuando en realidad es alta y pueda perjudicar al vendedor.

Ante estas posibles situaciones, el juez debería poder detectarlas y sancionar en el primer caso tanto al comprador como al vendedor, y en el segundo caso al comprador con relación a las normas implementadas.

Añadir sucesos externos que puedan perjudicar a los distintos agentes, es decir, situaciones que no puedan controlar y que perjudique a los productos que disponga y a los acuerdos pactados con otro agente.

Factores que afecten a los productos como pudiera ser, en el caso de lo implementado hasta ahora, enfermedades en los animales que pudieran echar a perder una partida de productos se podrían implementar en los propios productos o en los centros de producción.

Cada cierto tiempo, se podría hacer que aleatoriamente los productos de un agente de la cadena sufrieran una enfermedad y por lo tanto murieran (los productos quedan inservibles para la venta). Esto podría ocasionar en el agente que posee estos productos que pierda el dinero que invirtió para hacerse con ellos (pudiendo incluso arruinarse) y además que pueda perder reputación respecto a los compradores si ya tenía pactados dichos productos con un comprador.

Esto podría hacerse de la misma manera, si hubiera otro tipo de productos en la cadena de distribución y que también se pudieran echar a perder por motivos de fenómenos meteorológicos o por cualquier motivo ajeno al del agente propietario de dichos productos.

Factores que afecten a la hora de entregar los productos en una transacción realizada entre dos agentes.

Cuando dos agentes han cerrado una operación de compraventa de un producto el comprador paga al vendedor y este le entrega los productos. Se podría hacer que cada cierto tiempo, en una transacción ocurriera un suceso imprevisto que retarde la entrega de los productos, como pudiera ser un problema con el transporte de los productos por alguna causa. Este suceso podría suponer que su reputación bajara.

Añadir nuevo comportamiento al juez tanto para sancionar las nuevas acciones de los agentes y añadir más reglas de obligado cumplimiento que simulen las existentes en los distintos mercados de Mercamadrid.

Como puede verse, esta aplicación esta diseñada de tal manera que sea fácilmente extensible para futuros proyectos ya que debido a la complejidad del problema a tratar queda mucho por hacer para realizar mejores simulaciones.

Bibliografía

[1] George Akerlof (1970) “The Market for Lemons: Quality Uncertainty and the Market Mechanism” http://en.wikipedia.org/wiki/The_Market_for_Lemons http://ideas.repec.org/a/tpr/qjecon/v84y1970i3p488-500.html

[2] Mercados Centrales de Abastecimiento de Madrid MercaMadrid http://www.mercamadrid.es/

[3] Andreas Diekmann, Wojtek Przepiorka (2005) “The Evolution of Trust and Reputation: Results from Simulation Experiments” http://ideas.repec.org/p/wpa/wuwpex/0508005.html

[4] Jordi Sabater i Mir (2002) “Trust and reputation for agent societies” http://www.tdx.cesca.es/TESIS_UAB/AVAILABLE/TDX-0123104-172828//jsm1de1.pdf

[5] María Victoria Belmonte, Vicente J. Botti, Juan Manuel Corchado, Alberto Fernández, Ana García Serrano, Francisco Garito, Adriana Giret, Jorge Gómez Sanz, Josefa Z. Hernández, Vicente Julián, José Manuel Molina, Antonio Moreno, Pablo Noriega, Sascha Ossowski, Juan Pavón, Juan A. Rodríguez-Aguilar, Sergio Saugar, Carles Sierra (2005) “Agentes software y sistemas multi-agente: conceptos, arquitecturas y aplicaciones”

[6] Editado por Jeffrey M. BradShaw (1997) “Software Agents”

[7] Gul A. Agha, Les Passer, Kathleen M. Carley, Michael P. Georgeff, Jose Cuena, Michael N. Huhns, Edmund H. Durfee, Toru Ishida, Clarence (Skip) Elis, Nadeem Jamali, Sascha Ossowski, Larry N. Stephens, H. Van Dyke Parunak, Gerard Tel, Anand S. Rao, Jacques Wainer, Tuomas Sandholm, Gerhard Weiss, Sandip Sen, Mike Wooldridge, Munindar P. Singh, Makoto Yokoo (1999) Multiagent Systems. A Modern Approach to Distributed Artificial Intelligence.

[8] Reglamento de Funcionamiento del Mercado Central de Carnes - Boletín Oficial. Comunidad de Madrid 12/10/1985 pág. 37-42 http://www.munimadrid.es/portal/site/munimadrid/menuitem.8522c2ac1498b7f5b1e77112b95286a0/?vgnextoid=145c9d2e3fd4f010VgnVCM1000009b25680aRCRD&vgnextchannel=00dfb351fd18d010VgnVCM1000009b25680aRCRD

[9] Reglamento de Funcionamiento del Mercado Central de Pescados - Boletín Oficial. Comunidad de Madrid 01/07/1985 pág. 22-30

http://www.munimadrid.es/portal/site/munimadrid/menuitem.8522c2ac1498b7f5b1e77112b95286a0/?vgnextoid=2ffc9d2e3fd4f010VgnVCM1000009b25680aRCRD&vgnextchannel=00dfb351fd18d010VgnVCM1000009b25680aRCRD

[10] Reglamento de Funcionamiento del Mercado Central de Frutas Y Verduras - Boletín Oficial. Comunidad de Madrid 05/07/1985 pág. 19-26
http://www.munimadrid.es/portal/site/munimadrid/menuitem.8522c2ac1498b7f5b1e77112b95286a0/?vgnextoid=c71d9d2e3fd4f010VgnVCM1000009b25680aRCRD&vgnextchannel=00dfb351fd18d010VgnVCM1000009b25680aRCRD

[11] FIPA TC Communication (2002) FIPA Contract Net Interaction Protocol Specification http://www.fipa.org/specs/fipa00029/SC00029H.pdf

[12] FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS www.fipa.org

[13] Eclipse Homepage. http://www.eclipse.org/

[14] Java Homepage. http://www.java.com/
[15] Bouml Homepage. http://bouml.free.fr/

[16] Instituto Nacional de Estadística http://www.ine.es

[17] Sección Estadísticas de MercaMadrid
http://www.mercamadrid.es/es/estadisticas.html

[18] Andreas Schlosser, Marco Voss and Lars Brückner (2006) “On the Simulation of Global Reputation Systems”
http://jasss.soc.surrey.ac.uk/9/1/4.html

[19] RePast Homepage. http://repast.sourceforge.net.

[20] Wikipedia - English Auction http://en.wikipedia.org/wiki/English_auction

[21] Leonora Millán (2006) “Teoría de Subastas”
http://socioeconomia.univalle.edu.co/nuevo/public/index.php?seccion=DOCUMENTOS&download=1&documento=73&PHPSESSID=77347357a2420d613c85a16077a69ea6

[22] Article about Software Agent in www.wikipedia.org
http://en.wikipedia.org/wiki/Software_agent

[23] Trabajo sobre "Agent Reputation Trust (ART). Testbed Competition" 2006 Iván García-Magariño García. Agentes Inteligentes

[24] Lik Mui, Mojdeh Mohtashemi, Ari Halberstadt (2002) "A Computational Model Of Trust And Reputation" http://www.fi.muni.cz/~xvyborny/DMsouhrn/papers/reputation/reputace%20od%20marka/_A_COMPUTATIONAL_MODEL_OF_T.PDF

[25] Yao Wang, Julita Vassileva "Trust and Reputation Model in Peer-to-Peer Networks" http://www.cs.usask.ca/grads/yaw181/publications/120_wang_y.pdf

[26] Karen K. Fullam, Tomas B. Klos, Guillaume Muller, Jordi Sabater, Andreas Schlosser, Zvi Topol, K. Suzanne Barber, Jeffrey S. Rosenschein, Laurent Vercouter, Marco Voss "The Agent Reputation and Trust (ART) Testbed" http://www.lips.utexas.edu/art-testbed/pdf/SpecSummary.pdf http://www.lips.utexas.edu/art-testbed/

Anexo

1. Breve manual introductorio de usuario.

Para iniciar el programa se debe ejecutar el método main de la clase GUI, para ello se debe llamar a la máquina virtual de java con el comando "java".

Una vez ejecutado, aparecerán 3 ventanas, dos de ellas de RePast y el formulario principal.

La simulación debe iniciarse siempre desde ese formulario principal y nunca pulsando el botón play del formulario de RePast.

Para comenzar una simulación se debe pulsar en el botón "Iniciar" del menú "Mercado". Al hacer esto aparecerá un formulario de configuración que nos ofrece tres opciones para configurar la simulación.

- Configuración por defecto.

  Inicia la simulación con valores por defecto. (10 productores, 10 intermediarios, 10 vendedores de Mercamadrid y 10 consumidores todos ellos con características por defecto).

- Configuración aleatoria.

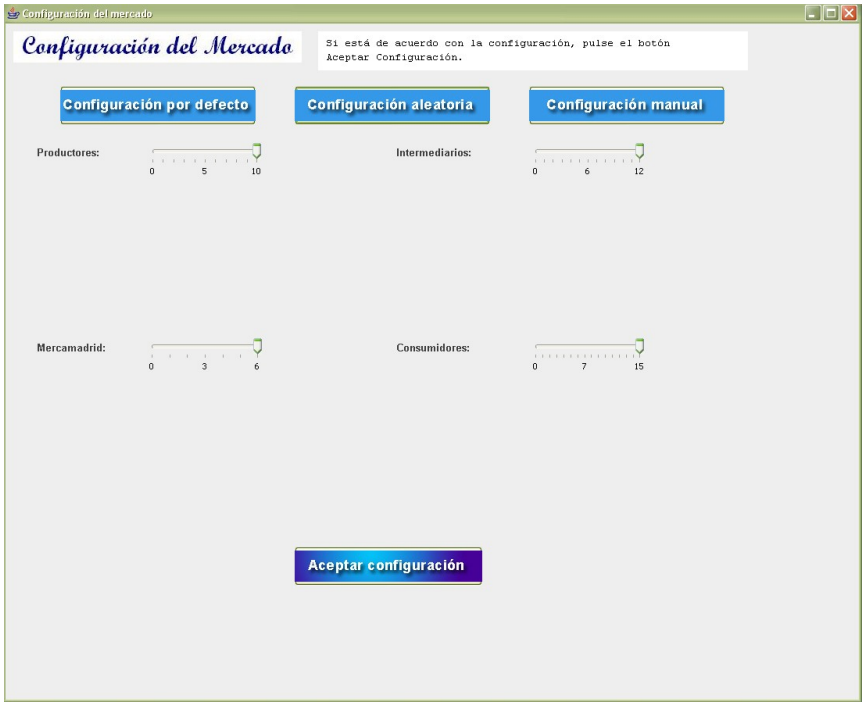
  Inicia la simulación con valores aleatorios.

- Configuración manual.

  Permite configurar manualmente los valores de la simulación.

  Se deben introducir primero el número de agentes que se desean de cada tipo (la cadena está preestablecida como la descrita en el apartado de adaptación) y después pulsar ENTER.

  Por cada nuevo valor que se modifique se debe pulsar la tecla ENTER para que se asigne.



Una vez iniciada la simulación ya se podrán utilizar los controles de RePast para pausarla, pararla o reanudarla y la interfaz gráfica para ver los valores de las variables.

La interfaz gráfica posibilita ver datos generales de los mercados, pulsando en sus fotografías, y datos de los agentes de cada mercado pulsando en el nombre del mercado y posteriormente en cada uno de los agentes.

Los agentes vienen separados en compradores y vendedores aunque en ocasiones se refieran a un solo agente. En la parte comprador vendrán sus características y valores como tal y en la parte vendedores sus propiedades como vendedor.

En los modos de configuración aleatorio y por defecto las gráficas de resultados aparecerán al cabo de 5000 unidades de tiempo y mostrarán valores tomados cada 10 unidades de tiempo. En el modo manual estos valores se pueden configurar.

En esta versión se muestran gráficas de dinero, ventas, confianza y reputación.

En los modos aleatorio y por defecto el juez de Mercamadrid está activado, afectando consecuentemente a las gráficas. En el modo manual se puede elegir si activarlo o no.

Es el modo manual el que se debe utilizar para recrear los experimentos descritos en esta memoria en el apartado de experimentación.

2. Traducción de un tutorial de uso de Repast.

Se adjunta en el CD de la aplicación.